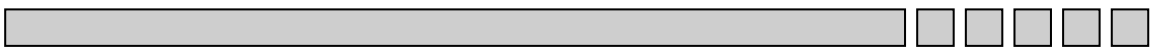
Intel Virtual Interface (VI) Architecture Conformance Suite User's Guide

Preliminary Version V0.5
December 14, 1998



DISCLAIMERS

THIS INTEL VIRTUAL INTERFACE ARCHITECTURE CONFORMANCE SUITE USER'S GUIDE ('USER'S GUIDE') IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR ANY OTHER RIGHTS OF THIRD PARTIES OR OF INTEL, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY SPECIFICATION, DOCUMENTATION, SOFTWARE OR OTHER MATERIALS REFERENCED HEREIN.

Nothing in this document constitutes a guarantee, warranty or license to any intellectual property right, express or implied, by estoppel or otherwise. Intel makes no representations or warranties and specifically disclaims all liability as to the User's Guide, the test suite software ('Software'), or specifications and the information and test cases contained therein with respect to: (i) liability for infringement of any proprietary rights, including without limitation, intellectual property rights; (ii) sufficiency, reliability, accuracy, completeness or usefulness of same; and (iii) ability or sufficiency of same to function accurately as a representation of any standard. Furthermore, Intel makes no commitment to update the information contained in the foregoing items, and Intel reserves the right to make changes at any time, without notice, to the User's Guide, the Software, the test cases, or the test specification.

Third parties may have intellectual property rights which may be relevant to this document and the technologies discussed herein, accordingly the reader is advised to seek the advise of competent counsel as required, without obligation to Intel.

SOFTWARE LICENSE AGREEMENT

BY INSTALLING OR USING THIS SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. DO NOT INSTALL OR USE UNTIL YOU HAVE CAREFULLY READ AND AGREED TO THE FOLLOWING TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ANY ACCOMPANYING ITEMS.

LICENSE. Intel Corporation ("Intel") grants You the nonexclusive, royalty-free right to install and use the Intel Virtual Interface Architecture Conformance Suite software programs ("Software"), pursuant to the terms and conditions set forth below:

YOU MAY:

1. Install, use, and copy the Software.
2. Make a reasonable number of backup and/or archival copies of the Software.
3. Publish the results derived from such use ("Results") and use, reproduce and distribute such Results pursuant to the following limitations: (i) The Results derived from each utilization of the Software (i.e., the "Test Report") must be published in its entirety, (ii) Each Test Report must include the complete disclaimers set forth in Exhibit A hereto, and (iii) Each Test Report must reference: (1) the specific version of the Software from which it was derived, (2) the conformance or lack of conformance test results, and (3) if a conformance test result is noted, the level of conformance must be specifically identified pursuant to Section 1.3 of the Intel Virtual Interface Architecture Conformance Suite User's Guide ("User's Guide"): Early Adopter, Functional, or Full Conformance.

YOU MAY NOT:

1. Copy, modify, rent, sell, distribute or transfer any part of the Software except as provided in this Agreement, and You agree to prevent unauthorized copying of the Software.
2. Reverse engineer, decompile, or disassemble the Software.
3. Sublicense the Software.

OWNERSHIP OF SOFTWARE AND COPYRIGHTS. Title to all copies of the Software remains with Intel or its suppliers. The Software is copyrighted and protected by the laws of the United States and other countries, and international treaty provisions. You may not remove any copyright notices from the Software. Intel may make changes to the Software, or to items referenced therein including without limitation, the test specifications, at any time without notice, but is not obligated to support or update the Software. Except as otherwise expressly provided, Intel grants no express or implied right under Intel patents, copyrights, trademarks, or other intellectual property rights. You may transfer the Software only if the recipient agrees to be fully bound by these terms and if You retain no copies of the Software.

THIRD PARTY SOFTWARE. Third party software (e.g., drivers, utilities, operating system components, etc.) which You may use in connection with use of the licensed Software is subject to the third party licenses supplied with such software. Intel expressly disclaims liability of any kind with respect to Your installation or use of third party software.

LIMITED MEDIA WARRANTY. If the Software has been delivered by Intel on physical media, Intel warrants the media to be free from material physical defects for a period of ninety days after delivery by Intel. If such a defect is found, return the media to Intel for replacement or alternate delivery of the Software as Intel may select.

EXCLUSION OF OTHER WARRANTIES. EXCEPT AS PROVIDED ABOVE, THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING, WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF PROPRIETARY OR INTELLECTUAL PROPERTY RIGHTS, FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF THE SPECIFICATION. Intel does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the Software.

LIMITATION OF LIABILITY. IN NO EVENT SHALL INTEL OR ITS SUPPLIERS BE LIABLE TO ANY PARTY FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, LOST PROFITS, BUSINESS INTERRUPTION, COMPUTER FAILURE OR MALFUNCTION, OR LOST INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. INTEL DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS CONCERNING (I) THE USE OF THE SOFTWARE OR ANY ACCOMPANYING WRITTEN MATERIALS, INCLUDING THE USER'S GUIDE, IN TERMS OF SUFFICIENCY, ACCURACY, RELIABILITY, COMPLETENESS OR USEFULNESS OF SAME, AND/OR THE ABILITY OR SUFFICIENCY OF SAME TO FUNCTION ACCURATELY AS A REPRESENTATION OF ANY STANDARD, (II) THE

USE OF, DISTRIBUTION OF OR RELIANCE UPON THE TEST REPORTS AND/OR THE RESULTS, OR (III) THE VIRTUAL INTERFACE ARCHITECTURE SPECIFICATION, THE INTEL VIRTUAL INTERFACE ARCHITECTURE DEVELOPER'S GUIDE, USER'S GUIDE, OR ANY OTHER MATERIALS RELATED TO THE IMPLEMENTATION OF THE SOFTWARE OR INTERPRETATION OF RESULTS. YOU REMAIN SOLELY RESPONSIBLE FOR THE DESIGN, SALE AND FUNCTIONALITY OF YOUR PRODUCT(S), INCLUDING, WITHOUT LIMITATION, ANY LIABILITY ARISING FROM PRODUCT LIABILITY OR PRODUCT INFRINGEMENT. SOME JURISDICTIONS PROHIBIT EXCLUSION OR LIMITATION OF LIABILITY FOR IMPLIED WARRANTIES OR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM JURISDICTION TO JURISDICTION.

TERMINATION OF THIS AGREEMENT. Intel may terminate this Agreement at any time if You violate its terms. Upon termination, You will immediately Discontinue use of the Software and destroy any copies of the Software or return all copies of the Software to Intel. Appropriate provisions shall survive termination of this Agreement.

APPLICABLE LAWS. Claims arising under this Agreement shall be governed by the laws of California, excluding its principles of conflict of laws and the United Nations Convention on Contracts for the Sale of Goods. You may not export the Software in violation of applicable export laws and regulations. Intel is not obligated under any other agreements unless they are in writing and signed by an authorized representative of Intel.

GOVERNMENT RESTRICTED RIGHTS. The Software is provided with "RESTRICTED RIGHTS." Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 *et seq.* or its successor. Use of the Software by the Government constitutes acknowledgement of Intel's proprietary rights therein. Contractor or Manufacturer is Intel Corporation, 2200 Mission College Blvd., Santa Clara, CA 95052.

SEVERABILITY. The terms and conditions stated in this Agreement are declared to be severable. If any paragraph, provision, or clause in this Agreement shall be found or held to be invalid or unenforceable in any jurisdiction in which this Agreement is being performed, the remainder of this Agreement shall remain valid and enforceable.

COMPLETE AGREEMENT. This is the complete any exclusive statement of this Agreement between You and Intel. This Agreement supersedes any and all prior agreement or understandings, written or oral, with respect to the subject matter of this Agreement.

EXHIBIT A

THIS TEST REPORT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR ANY OTHER RIGHTS OF THIRD PARTIES OR OF INTEL, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY SPECIFICATION, DOCUMENTATION, SOFTWARE OR OTHER MATERIALS REFERENCED HEREIN.

Nothing in this document constitutes a guarantee, warranty or license to any intellectual property right, express or implied, by estoppel or otherwise. Intel makes no representations or warranties and specifically disclaims all liability as to the Test Report, the test suite software ('Software') which generated the Test Report, specifications referenced thereby, or information and test cases contained therein, with respect to: (i) liability for infringement of any proprietary rights, including without limitation, intellectual property rights; (ii) sufficiency, reliability, accuracy, completeness or usefulness of same; and (iii) ability or sufficiency of same to function accurately as a representation of any standard. Furthermore, Intel makes no commitment to update the information contained in any of the foregoing items, and Intel reserves the right to make changes at any time, without notice, and without limitation, to the Software, the test cases, or the test specification.

Third parties may have intellectual property rights which may be relevant to this document and the technologies referenced herein, accordingly the reader is advised to seek the advise of competent counsel as required, without obligation to Intel.

Table of Contents

1. Introduction

1.1. Overview

The *Virtual Interface Architecture Specification* describes the architecture for a high bandwidth/low latency interface between high performance network hardware and computer systems. The *VI Architecture Specification* is primarily intended for network hardware vendors and operating system vendors, rather than application programmers.

In the interest of promoting rapid support for the VI Architecture and a uniform Application Program Interface (API) to VI, Intel has developed the *Intel Virtual Interface (VI) Architecture Developer's Guide*. It describes the Virtual Interface Provider Library (VIPL) and the VI Kernel Agent, which are based on the example VI User Agent in Appendix A of the *VI Architecture Specification*.

This Intel VI Architecture Conformance Suite is a tool intended to aid in the determination of conformance of a VIPL implementation to the VI specification, especially to the recommended interface in the *Intel Virtual Interface (VI) Architecture Developer's Guide*.

1.2. Related Documents

Information on the Virtual Interface Architecture is available from the following documents:

- *Virtual Interface Architecture Specification.* Compaq/Intel/Microsoft; Revision 1.0, December 16, 1997 (available at http://www.viarch.org/html/Spec/vi_specification_version_10.htm).
- *Intel Virtual Interface (VI) Architecture Developer's Guide.* Intel; Revision 1.0, September 9, 1998 (available at http://www.intel.com/design/servers/vi/developer/ia_imp_guide.htm).
- *Intel Virtual Interface (VI) Architecture Developer's Guide Error Table Supplement.* Intel; Revision 1.0, September 3, 1998 (at http://www.intel.com/design/servers/vi/developer/ia_imp_guide.htm).

1.3. Levels of Conformance

Conformance phases have been defined with input from Independent Software Vendors (ISVs) and consultation with Independent Hardware Vendors (IHVs) that are planning products developed to the VIPL interface. The ISV community can use the phases to determine which VI NIC products provide the functions needed for demonstrations and products and the VI NIC vendors can use this to understand the needs of the ISVs.

Three phases have been defined and have been named as follows: Early Adopter, Functional and Full Conformance. The Early Adopter phase has been broadly defined as the set of VIPL API calls and other functionality in the API required for demonstrations and application prototyping. The Functional phase contains the additional functions that ISVs plan to use to develop products. Finally, Full Conformance is defined to be the full set of API and functions defined in *the Intel VI Architecture Developer's Guide.*

Details of the requirements for each of these conformance phases are contained in the following sections.

1.3.1. Early Adopter

The following are the calls and related functionality required to meet the requirements of the Early Adopter phase:

VipOpenNic()
VipCloseNic()

VipCreateVi()
VipDestroyVi()

VipConnectWait()
VipConnectAccept()
VipConnectReject()
VipConnectRequest()
VipDisconnect()

VipRegisterMem()
VipDeregisterMem()

VipPostSend()
VipSendDone()
VipSendWait()
VipPostRecv()
VipRecvDone()
VipRecvWait()

VipQueryNic()
VipQueryVi()
VipQueryMem()

Additional functionality requirements: Reliable Delivery and RDMA Write (although Reliable Delivery is not a requirement of the *VI Architecture Specification*, ISVs have stated they require Reliable Delivery).

Note: It is acceptable to ignore the memory protection tags in the memory protection checks for the early adopter release. See Section 1.5.2 for information on how this is handled by the conformance tests.

1.3.2. Functional

Functional conformance includes the requirements of Early Adopter above, plus:

VipConnectPeerRequest()
VipConnectPeerDone()
VipConnectPeerWait()

VipCQDone()
VipCQWait()
VipCreateCQ()
VipDestroyCQ()
VipResizeCQ()

VipCreatePtag()
VipDestroyPtag()

VipNSInit()
VipNSGetHostByName()
VipNSGetHostByAddr()
VipNSShutdown()

VipErrorCallback()

Additional functionality requirements: Protection Tag support.

If supported by the VIPL implementation, RDMA Read will be tested at this level of conformance.

1.3.3. Full Conformance

Full conformance is defined to be full support of the VI Architecture, which includes the remaining functions:

VipSendNotify()
VipRecvNotify()
VipCQNotify()

VipQuerySystemManagementInfo()

VipSetViAttributes()
VipSetMemAttributes()
VipQueryMem()

Additional Requirements: Unreliable Delivery and (optionally) Reliable Reception.

1.4. System Requirements

1.4.1. Hardware

Two or more IA systems, each containing:

- VI NIC,
- Secondary MS-NET (NT LAN Manager and TCP/IP) interconnect.

1.4.2. Software

Each system in the cluster to be tested must contain:

- Microsoft* Windows NT Version 4.0 + Service Pack 3,
- TCP/IP networking installed and configured,
- RSHD (Rsh service daemon),
- VI NIC run-time software (DLL files).

The system running the test harness must also contain:

- NMAKE and C compiler (to process makefiles and compile Win32 C source files),
- Perl interpreter for NT,
- VI NIC development software (vipl.h and vipl.lib).

Additional information about the C compiler, Perl interpreter and RSHD service that were used to develop and test the Conformance Suite is in Appendix A.

1.5. Limits and Assumptions

1.5.1. Scalability

A minimum of two nodes is required for most tests that make connections. Tests are designed to scale with the number of nodes in the cluster.

Conformance Suite tests are designed for clusters of up to 16 nodes, but have only been tested on up to four nodes. In a system with multiple NICs, only one NIC will be tested at a time.

1.5.2. Early Adopter Conformance

All tests that make connections make calls to *VipCreatePtag()*, *VipDestroyPtag()* and *VipErrorCallback()*, which are not required for Early Adopter phase conformance. If the implementation under test does not support these calls, then "stubs" for these calls, which return VIP_SUCCESS, must be provided.

1.5.3. Unreliable Delivery

Sends and RDMA writes may be lost on an Unreliable VI. The conformance tests assume that unreliable data will be delivered, and will fail if an expected message is not delivered. The algorithms of these tests are simple enough that data loss should not occur, and there is no way to detect if an error is due to non-conformance or due to unreliability.

1.5.4. Resource Limits Testing

The NIC attributes returned by the *VipQueryNic()* function are not dynamically updated. Limits tests assume that these values do not include resources consumed by the VIPL implementation itself, and that these resources are all available to the test programs. If other test cases or vendor-supplied applications (such as a VIPL NDIS driver) are running simultaneously with limits tests, non-conformance will be flagged by limits tests.

The systems being tested should also be configured enough memory in order to allocate the limits returned by the *VipQueryNic()* call, or some limits tests will fail.

1.5.5. Negative Tests

The tests in the error and async directories test how an implementation handles programmer errors. Some of these tests produce errors by passing invalid handles or null pointers. These tests will "pass" if the correct VIP_RETURN code is returned, or "verify" if a program exception occurs. The VI specification does not disallow an implementation from causing a program exception when a user passes an invalid value, so a "verify" response is not necessarily incorrect or unacceptable. On the other hand, if a negative test causes system errors that prevent other applications from running, then this is clearly unacceptable.

2. Installation and Quick Start

- Install Windows NT, C compiler, Perl interpreter, RSHD service and NIC software. Make sure that the system environment variables PATH, LIB and INCLUDE reflect the installations correctly.

  *Additional information about the C compiler, Perl interpreter and RSHD service that were used to develop and test the Conformance Suite is in Appendix A. These are **not** supplied with the Conformance Suite.*

- Install the Conformance Suite on a server drive accessible to each VI test system. The files can be installed on either a shared drive on one of the test systems, or on another server that can be mapped by each system in the cluster to be tested.

  Run vipconf5.exe and follow the instructions in the program. It will install the test cases and harness, and set the system environment variable VIPCONF_HOME to the "root" of the Conformance Suite tree. While it is not required that you reboot the system after installing the Conformance Tests, note that required environment variables will not be updated until your next login.

- Edit the file %VIPCONF_HOME%\bin\netaddr to contain the VI NIC address followed by the NT LAN Manager network name of each test system in the cluster, e.g.:

```
00.aa.00.12.34.56          system1
00.aa.00.78.9a.bc          system2
```

  See Section 3.2 for information on using VIPL Name Service names instead of physical NIC addresses.

- Edit %VIPCONF_HOME%\include\vipconf.h as necessary (see directions in the file).

  At a minimum, you will need to set the NIC Name and level of conformance here, but check the other values as well. See Section 1.3 for more details on levels of conformance.

- Run the Conformance Suite, in a Command Prompt window, e.g.:

```
cd %VIPCONF_HOME%
perl bin\Vipconf.pl /t testlist<n>
```

  where <n> is the number 1, 2 or 3, corresponding to Early Adopter, Functional and Full Conformance respectively. See Section 1.5.2 for a special note regarding assumptions for Early Adopter conformance.

  See Section 3.3 for details on running the Conformance Suite.

- To uninstall the Conformance Suite, you must first remove all the generated files (logs, results, executables and other intermediate files) with:

```
cd %VIPCONF_HOME%
perl bin\Vipconf.pl /t testlist3 /uninstall
```

  Then, uninstall the Conformance Suite with the Control Panel's Add/Remove Programs icon.

  Select "Intel Virtual Interface Architecture Conformance Suite" and press Add/Remove.

3. Test Harness

3.1. Overview

The VI Conformance Suite is Perl script driven. The file %VIPCONF_HOME%\bin\Vipconf.pl is the Perl script responsible for establishing the test environment, building the tests, and then running the test cases on all nodes specified in the file %VIPCONF_HOME%\bin\netaddr. This work is done by the "master node".

3.2. Node Communication

A separate Network Interface Card configured with TCP/IP transport is used for test harness inter-node communication. This is done to avoid consuming VI resources in the systems under test, which may cause unexpected errors. The mechanism used for running test cases on cluster nodes is the RSH (remote shell) command, where each cluster node has mapped the shared drive containing the conformance test files with the NT LAN Manager NET USE commands.

The file %VIPCONF_HOME%\bin\netaddr is used by the harness to determine the network name of each cluster node, and by test cases to determine the NIC addresses of the other cluster nodes. This text file contains one entry for each cluster node, and must be configured by the user before running the test harness. Each line contains a VI NIC address followed by the NT LAN Manager network name of each test system in the cluster, e.g.:

```
00.aa.00.12.34.56          system1
00.aa.00.78.9a.bc          system2
```

If you are testing for Functional or Full Conformance, you may provide a VIPL Name Service name instead of a VI NIC address. The VIPL name must be enclosed in quotation marks, e.g.:

```
"ViplTest1"                system1
"system2"                  system2
```

When specified in this format, tests will use *VipNSInit()* and *VipNSGetHostByName()* to obtain the default NIC address for the specified systems. Note that the VIPL Name Service name may or may not be the same as the NT LAN Manager network name; for that reason both must be supplied, as shown above.

3.3. Detailed Description

This section describes how the test harness works. The general algorithm is as follows:

Harness system "Master node"	Cluster node(s)
Build required libraries and tools	*Wait for work (RSH service)*
Share test directories (if %VIPCONF_HOME% is on a local file system) `net share . . .`	
Map test directories on all Cluster nodes	`net use . . .`
For each test case: (either a specified text file list or the test in the current directory if not specified)	
Chdir to test directory	

Issue nmake to build test *log.nmake*	
Invoke test remotely, via RSH, on each Cluster node, and locally if harness is executing on a Cluster node.	Test executable invoked
Execute test and log test output *log.#* Generate local result (Pass/Fail/Verify) *result.#*	Execute test and log test output *log.#* Generate local result (Pass/Fail/Verify) *result.#*
Synchronize with all Cluster nodes	
Analyze results from each Cluster node	*Wait for work (RSH service)*
Generate overall result for test *result*	
Until all tests run	
Generate Overall Pass/Fail report	

The "master node" runs the conformance tests within the harness framework. When Vipconf.pl is launched on the master node, it sets up the test environment, including:

- building the test libraries in %VIPCONF_HOME%\lib,
- building other binaries used by the harness in %VIPCONF_HOME%\bin,
- discovering the characteristics of the %VIPCONF_HOME% drive (i.e. is it a UNC mapped drive or a local drive, etc.),
- generating a list of host names in the cluster from %VIPCONF_HOME%\bin\netaddr, and mapping %VIPCONF_HOME% on the cluster nodes where it is not already mapped, and
- parsing the testlist.

Once the environment is set up, the harness builds a test using "nmake", and then launches the test on each node in the cluster via an "rsh" command. The test harness prints out various messages with time stamps to monitor the progress of the tests. After all the tests have completed, the results are tabulated and printed to the screen. See Section 4 for information on interpreting and investigating test failures.

An example of a small test run is shown below. *The lines in italics appear only if the Conformance Suite is installed on a file system local to the Master Node.*

```
P:\Vipconf>perl bin\Vipconf.pl /t testlist1 /home p:\vipconf
------- 04/28/1998  3:20p: Creating VIPCONF_HOME share -------
vipconf was shared successfully.
------- 04/28/1998  3:20p: Building vipconf.lib -------
------- 04/28/1998  3:20p: Building vipconf utilities -------
------- 04/28/1998  3:20p: Checking VIPCONF_HOME on remote nodes -------
------- 04/28/1998  3:20p: Checking remote node righty -------
System error 1326 has occurred.

Logon failure: unknown user name or bad password.
Remote UNC name not found on righty: sending 'net use' command
Please enter domain and username (domain\username): mydomain\myid
Please enter password:
------- 04/28/1998  3:20p: Checking remote node lefty -------
System error 1326 has occurred.

Logon failure: unknown user name or bad password.
Remote UNC name not found on lefty: sending 'net use' command
------- 04/28/1998  3:21p: Building ConnectWaitTO -------
```

```
------- 04/28/1998  3:21p: Starting test: ConnectWaitTO on host: righty -------
------- 04/28/1998  3:21p: Starting test: ConnectWaitTO on host: lefty -------
------- 04/28/1998  3:21p: Running test: ConnectWaitTO on host: righty -------
VIPCONF info  (1) Starting ConnectWait() Timeout Functional Test
VIPCONF result(1) ConnectWait() Timeout Functional Test all tests PASSED (13)
------- 04/28/1998  3:21p: Running test: ConnectWaitTO on host: lefty -------
VIPCONF info  (0) Starting ConnectWait() Timeout Functional Test
VIPCONF result(0) ConnectWait() Timeout Functional Test all tests PASSED (13)
------- 04/28/1998  3:21p: Gathering results: ConnectWaitTO -------
Test Result: P  (26 tests passed) ConnectWait() Timeout Functional Test
------- 04/28/1998  3:21p: Building ConnectRequestINF -------
------- 04/28/1998  3:21p: Starting test: ConnectRequestINF on host: righty -------
------- 04/28/1998  3:21p: Starting test: ConnectRequestINF on host: lefty -------
------- 04/28/1998  3:21p: Running test: ConnectRequestINF on host: righty -------
VIPCONF info  (1) Starting Connect Functional Test
VIPCONF result(1) Connect Functional Test all tests PASSED (14)
------- 04/28/1998  3:22p: Running test: ConnectRequestINF on host: lefty -------
VIPCONF info  (0) Starting Connect Functional Test
VIPCONF result(0) Connect Functional Test all tests PASSED (20)
------- 04/28/1998  3:22p: Gathering results: ConnectRequestINF -------
Test Result: P  (34 tests passed) Connect Functional Test
------- 04/28/1998  3:23p: All tests processed -------
------- 04/28/1998  3:23p: Closing connections -------
\\machine\share$ was deleted successfully.
\\machine\share$ was deleted successfully.
vipconf was deleted successfully.
-----------------------------------------------------------
P       Pass
F       Fail
 V      Requires additional manual verification of result
U       Unknown result code
N       No results found in test directory
E       Error (no such test directory)
-----------------------------------------------------------

P       tests\basic\ConnectWaitTO
P       tests\basic\ConnectRequestINF

-----------------------------------------------------------
Passed:          2
Pass+Verify:     0
Failed:          0
Fail+Verify:     0
No results:      0
Unknown results: 0
Errors:          0
Total:           2
```

3.4. Test Harness Usage

The test harness is invoked as follows:

```
[perl] [drive:][%VIPCONF_HOME%\bin\]Vipconf.pl [/t <testlist>] [/home <dir>]
                [/debug <num>] [/relmask <num>] [/clean] [/build] [/uninstall]
```

The options are used as follows:

/t <testlist> is used to specify a list of tests to run. Three lists are supplied with the Conformance Suite, in %VIPCONF_HOME%, corresponding to tests which apply to each level of conformance:

testlist1 Early Adopters
testlist2 Functional
testlist3 Full Conformance

A testlist consists of a list of test directories relative to the installed location of the Conformance Suite %VIPCONF_HOME%. Users can create their own testlist to specify which tests are to be run.

NOTE: The test directories specified in the <testlist> file are case sensitive and must use a backslash (\) as a delimiter, i.e.:

Correct: tests\basic\InterfaceEA
Incorrect: tests\basic\interfaceea
Incorrect: tests/basic/InterfaceEA
Incorrect: P:\vipconf\tests\basic\InterfaceEA

NOTE: A test directory which begins with a # is treated as a comment.

If no testlist is supplied on the command line, then the default action is to run in the current directory, which may located anywhere in the directory tree under %VIPCONF_HOME%.

/home <dir> is used to specify the location of the Conformance Suite. This switch overrides the %VIPCONF_HOME% environment variable. The <dir> can be on a local drive, in which case the drive will be "shared" and used by all other nodes under test; or it can be on a mapped network drive, in which case all nodes under test will use the UNC name of the mapped drive to run tests. *It is the responsibility of the tester to make sure the drive is accessible to the other test nodes*.

It is recommended that the user set the %VIPCONF_HOME% environment variable on the "master node" for general testing. If the /home switch is not used, the test harness uses %VIPCONF_HOME% as the location of the Conformance Suite.

/debug <num> Causes the debug versions of the tests (testd.exe) to be built and run, and controls the detail of test output that is produced. <num> can be 0, 1 or 2; /debug 0 corresponds to no additional output, /debug 1 will print more information during tests, and /debug 2 will print a great deal more information during tests.

/relmask <num> Specifies the ReliabilityLevel(s) to test. Using 1 for Unreliable Delivery, 2 for Reliable Delivery, and 4 for Reliable Reception, add together the desired ReliabilityLevels to be tested. The value must be between 1 and 7, specifying only ReliabilityLevels that are actually supported by the NIC under test. The default depends on the level of conformance specified in the file %VIPCONF_HOME%\include\vipconf.h (2 for Early Adopters and Functional conformance; the value of the ReliabilityLevelSupport attribute value returned by the *VipQueryNic()* call for Full Conformance).

/clean tells the harness to rebuild the test harness libraries, utilities, and all individual tests from source. Test cases will still be built and executed.

/build is used to build but not run the tests. /build can be used in conjunction with /clean to clean and rebuild the test harness library and tests.

/uninstall is used to remove all generated files (logs, results, executables and other intermediate files). Test cases are not built or executed.

3.5. Test Harness Customization

The harness uses the output from the `net use` command to determine the success of mount operations. If the systems under test are configured for a language other than English, you may need to edit the harness. See the instructions in %VIPCONF_HOME%\bin\Vipconf.pl.

3.6. Results Summary Usage

A summary of results of each test run is printed at the end of a Vipconf.pl run. The same results summary can be obtained by running %VIPCONF_HOME%\bin\Result.pl:

```
[perl] [drive:][%VIPCONF_HOME%\bin\]Result.pl [/t <testlist>] [/home <dir>]
```

The options behave as follows:

/t <testlist> is used to specify a list of tests from which results are to be gathered. Three lists are supplied with the Conformance Suite, in %VIPCONF_HOME%, corresponding to tests which apply to each level of conformance:

testlist1	Early Adopters
testlist2	Functional
testlist3	Full Conformance

A testlist consists of a list of test directories relative to the installed location of the Conformance Suite %VIPCONF_HOME%. Users can create their own testlist to specify which tests are to be run.

NOTE: The test directories specified in the <testlist> file are case sensitive and must use a backslash (\) as a delimiter, i.e.:

Correct: tests\basic\InterfaceEA
Incorrect: tests\basic\interfaceea
Incorrect: tests/basic/InterfaceEA
Incorrect: P:\vipconf\tests\basic\InterfaceEA

NOTE: A test directory which begins with a # is treated as a comment.

If no testlist is supplied on the command line, then the default action is to report the results in the current directory.

/home <dir> is used to specify the location of the Conformance Suite. This switch overrides the %VIPCONF_HOME% environment variable.

3.7. Vipconf Directory Hierarchy

The VI Conformance Suite is installed in the following directory tree:

%VIPCONF_HOME%	\bin		*netaddr file, test harness, and other tools*
	\lib		*conformance suite library*
	\include		*include files*
	\tests	\async	*test1…testn*
		\basic	*test1…testn*
		\error	*test1…testn*
		\limit	*test1…testn*
		\stress	*test1…testn*
		\template	*(see Sec. 4.2.4.2)*

Test cases are maintained in the tree structure, with one test case per directory. Each test directory contains:

- Test source file (*test.c*), and
- NMAKE file (*test.mak*).

The following files are generated when each test case is invoked by the harness:

- Make log (*log.nmake*),
- Test executable (*test.exe or testd.exe, plus intermediate files*),
- Results log from each node (*log.#*),
- Results file from each node (*result.#*), and
- Overall test case result (*result*).

where # indicates the index of the NIC in the %VIPCONF_HOME%\bin\netaddr file, starting at 0.

4. Interpreting Test Results

4.1. Test Summary

The test summary screen lists the status of each test harness run. An example of the format is shown below:

```
------------------------------------------------------------
P       Pass
F       Fail
 V      Requires additional manual verification of result
U       Unknown result code
N       No results found in test directory
E       Error (no such test directory)
------------------------------------------------------------

P       tests\basic\ConnectWaitTO
P       tests\basic\ConnectRequestINF

------------------------------------------------------------
Passed:          2
Pass+Verify:     0
Failed:          0
Fail+Verify:     0
No results:      0
Unknown results: 0
Errors:          0
Total:           2
```

Each test case will report one the following statuses on completion:

Passed	Implementation was conformant.
Failed	Implementation was non-conformant, or the test was unable to run due to a configuration error (such as too few nodes configured to run the test).

In addition, a few tests may report an additional **V**erify status. This identifies items that the Conformance Suite could not automatically validate, and must therefore be manually verified by the user. For example:

- provider-specific information (such as the version returned by the *VipQueryNic()* call),
- non-conformance against actions that are "strongly recommended" but not required by the *VI Architecture Specification* or the *Intel VI Architecture Developer's Guide*.

Other possible result codes reported by the Results.pl script are:

Unknown	The results script did not understand the results (this usually indicates a test case that did not terminate properly).
No results	No results were found in test directory (test was not run, results were deleted, or this is not a test directory).
Error	No such directory.

4.2. Analyzing Test Results

The result of each test is contained in its directory in a file called "result". The test result will be one of:

- P
- F
- PV
- FV
- U

which correspond to the results shown by the summary script. An example of a result file is shown below:

```
P:\Vipconf\tests\basic\Example>type result
Test Result: FV (2 tests failed of 14, must verify 8) VipQueryNic() Example Test
P:\Vipconf\tests\basic\Example>
```

In this example, 2 cases failed out of a total of 14, and 8 cases have results that must be verified manually by the user. The following sections tell how to determine the reason for the failures.

4.2.1. Build Failures

If a test fails to build, a message will be printed to the screen at the time of the build failure, and the results summary information will indicate "No results found". The output of the build commands is captured in a file in each test directory called "log.nmake". This file can be examined to determine the cause of the build failure.

4.2.2. Test Failures

Test failures are printed to the screen at the time of execution, and the summary screen will report "Failed" for the test in question. Individual test logs are also generated, and can be found in each test directory with the name "log.*#*" where *#* is the index in the %VIPCONF_HOME%\bin\netaddr file of the cluster node which generated that log file. Errors will report the line number of the test source where the failure occurred, and will report the error code when applicable. An example of a log file is shown below.

```
P:\Vipconf\tests\basic\Example>type log.0
VIPCONF info  (0) Starting VipQueryNic() Example Test
VIPCONF info  (0) Please verify the following NIC Attributes:
VIPCONF verify(0)  HardwareVersion=1
VIPCONF verify(0)  ProviderVersion=6512
VIPCONF FAILED(0)  MaxDiscriminatorLen=12, must be >=16 (test.c line 163)
VIPCONF verify(0)  ReliabilityLevelSupport=2
VIPCONF info  (0)                          (VIP_SERVICE_RELIABLE_DELIVERY)
VIPCONF verify(0)  RDMAReadSupport=0
VIPCONF result(0) VipQueryNic() Example Test 1 tests FAILED, 4 must manually verify (of 7)
P:\Vipconf\tests\basic\Example>
```

Messages from a test case will be prefaced with the string "`VIPCONF` *<message type>* (*#*) ", where *#* is the node number and *<message type>* is one of:

`info`	Informational messages.
`verify`	Identifies information that must be manually verified.
`FAILED`	Indicates non-conformance detected. Also includes the name of the file and source line number.

`result` The final result. Also includes the name of the file and source line number if the test is terminating due to a fatal error.

`debug1` Additional information printed only with the `/debug 1` or `/debug 2` switch.

`debug2` Additional information printed only with the `/debug 2` switch.

In the example above, the test case failed because the maximum discriminator length NIC attribute was too small. There are also four other values that the test found to be conforming, but could not determine if they were correct.

Further information on individual test cases is contained in Appendix B, Test Case Summary, as well as the source files for each individual test.

Since this test was run on two nodes, there will also be a log.1 file. When a test fails, all logs should be examined, since it is likely that different nodes will have different log files.

4.2.3. Re-running Test Cases Manually

In some cases, it may be desirable to run individual test cases manually without the harness. To do so, cd to the appropriate test directory on each system and invoke the test with:

```
test[d].exe [/home <dir>] [/debug <num>] [/relmask <num>] [/case <num>]
```

The options are used as follows:

/home <dir> is used to specify the location of the Conformance Suite. This switch overrides the %VIPCONF_HOME% environment variable.

> NOTE: %VIPCONF_HOME% is only set on the "master" node when the Conformance Suite is installed. It is supplied by the harness when tests are invoked on remote nodes, and must be supplied by the user when running manually.

/debug <num> causes the tests to produce output that is more detailed. <num> can be 0, 1 or 2. /debug 0 is the default; /debug 1 will print more information during tests; and /debug 2 will print a great deal more information during tests.

/relmask <num> specifies the ReliabilityLevel(s) to test. Using 1 for Unreliable Delivery, 2 for Reliable Delivery, and 4 for Reliable Reception, add together the desired ReliabilityLevels to be tested. The value should be between 1 and 7, specifying only ReliabilityLevels that are actually supported by the NIC under test. Default depends on the level of conformance specified in the file %VIPCONF_HOME%\include\vipconf.h (2 for Early Adopters and Functional conformance; the value of the ReliabilityLevelSupport attribute value returned by the *VipQueryNic()* for Full Conformance).

/case <num> for tests with multiple cases, this specifies the specific case to run. The default is –1, which indicates run all cases.

You may specify test.exe, the Release configuration binary, or testd.exe, the Debug configuration binary.

4.2.4. Building Test Cases Manually

4.2.4.1. Building Tests With "nmake"

Should the need arise to compile a test case outside the harness environment, the following environment should be established:

- The INCLUDE environment variable should include the directory path containing the Conformance Suite include file vipconf.h, which is in %VIPCONF_HOME%\include, and the path containing the VI Provider Library include file vipl.h.
- The LIB environment variable should include the path containing the Conformance Suite library vipconf.lib, which is in %VIPCONF_HOME%\lib, and the path containing the VI Provider Library vipl.lib.

In a Command Prompt window, change the present working directory to the test directory. To build a release version of the test (test.exe), invoke nmake as follows:

```
nmake /f test.mak
```

To build a debug version of the test (testd.exe), use:

```
nmake /f test.mak CFG="test – Win32 Debug"
```

The conformance test libraries, vipconf.lib or vipconfd.lib, in %VIPCONF_HOME%\lib are built with:

```
nmake /f lib.mak
```

or

```
nmake /f lib.mak CFG="lib – Win32 Debug"
```

for the release and debug versions respectively.

Refer to Section 4.2.3 for the proper syntax to invoke the test case manually on each node.

4.2.4.2. Building Tests in the Developer Studio Environment

A generic Microsoft* Developer Studio workspace that can be used to build any test in the Conformance Suite has been provided in the directory %VIPCONF_HOME%\tests\template. To use the generic workspace, copy the files test.dsw and test.dsp to the desired test directory and open the Developer Studio workspace file test.dsw.

The following Developer Studio settings must be present in order to build a test case:

- In the Tools->Options->Directories Tab, %VIPCONF_HOME%\include and the path containing the VI Provider Library include file vipl.h must be added to the "Include Files" directory list.
- In the Tools->Options->Directories Tab, %VIPCONF_HOME%\lib and the path containing the VI Provider Library vipl.lib must be added to the "Library Files" directory list.

Refer to Section 4.2.3 for the proper syntax to invoke the test case manually on each node.

Appendix A. Vipconf Environment Tested

The Conformance Suite was developed and tested using the following tools:

- **C compiler**

  Test cases and binary components of the harness were built and tested with Microsoft* Visual C++ V5.0. Visual C++ is *not* supplied with the conformance tests.

- **Perl**

  Harness scripts were developed and tested using ActiveState Perl for Win32, Build 315. Perl is *not* supplied with the conformance tests; it may be obtained, subject to the GNU General Public License, at http://www.activestate.com.

- **RSH service daemon**

  The test harness uses the Windows NT RSH.EXE command on the harness master node to execute test programs on remote nodes. The Conformance Suite was developed and tested using the Remote Shell Daemon for Windows NT/95 Version 1.6 on the other cluster nodes, which is archived at the Swedish University Network (SUNET) FTP archive. RSHD is *not* supplied with the conformance tests; it may be obtained at http://ftp.sunet.se/ftp/pub/pc/Windows95/mirror-indstate/Daemons/RSHD/.

  The servers used for testing were all members of the same NT workgroup. The RSHD service was installed and started as the "Administrator" user, with an entry for the "master" node in the ".rhosts" file, on each server. Reference the *Remote Shell Daemon for Windows NT/95 Version 1.6* documentation for further details.

Appendix B. Test Case Summary

This section is modified as test cases are completed and debugged; however change bars are turned off in this section.

Legend:

Type:
Directory containing test under %VIPCONF_HOME%\tests.

TL:
Test Level; the *minimum* level of conformance required:
1 - Early Adopters.
2 - Functional.
3 - Full Conformance.

Name:
Test case name. For further information on test cases, refer to the indicated source file.

Description:
Short description of test and expected results.

Note that (NULL *) or other invalid handle values as an erroneous input parameter may not produce the expected result listed; however it is tested to ensure the error condition does not cause harmful effects on the system. The non-portable __try{} / __except{} construct is used to trap this condition.

Basic tests that specify an unreasonably large combination of parameters (such as testing from 0..NicAttr.MaxTransferSize bytes on connection establishment or data transfer) have been implemented so that the default will be the limits and a reasonable subset of values in between.

Section:
Reference section in the *VI Architecture Specification (AS:)* and/or the *Intel VI Architecture Developer's Guide (DG:)*.

AS:x.x.x +	DG:x.x.x	Indicates that the *Intel VI Architecture Developer's Guide* contains additional information that supplements the *VI Architecture Specification*.
AS:x.x.x /	DG:x.x.x	Denotes information that is identical in both documents.
AS:x.x.x		Denotes information contained only in the *VI Architecture Specification*.
	DG:x.x.x	Denotes information contained only in the *Intel VI Architecture Developer's Guide*.

Type	TL	Name	Description	Section		
basic	1	**InterfaceEA**	Test includes reference to all interfaces, data structures and constants required for Early Adopter conformance. *Intended as 1-stop test to identify if there are any incompatibilities that will prevent conformance tests from compiling or linking, as well as test the Vipconf library functions used by all tests.* *If the test compiles and links with the VI library, then the test will pass if the test harness is correctly configured, the program can determine it's NIC address, and the #define constants and enumerated values are correct. If not, part (or all) of the conformance suite will not be usable unless adjustments are made by the end user.*	AS:9.*	+	DG:2.*, 3.* *DG:2.2.1*
basic	2	**InterfaceFunc**	Test includes reference to all interfaces, data structures and constants required for Functional conformance. *Same as above, with added interfaces (peer to peer connection, completion queues, and name service).*	AS:9.*	+	DG:2.*, 3.* *DG:2.2.2*
basic	3	**Interface**	Test includes reference to all interfaces, data structures and constants required for Full conformance. *Same as above, with all interfaces (*Notify, other misc. functions).*	AS:9.*	+	DG:2.*, 3.* *DG:2.2.3*

interface.xls

B.1. Hardware Connection

Type	TL	Name	Description	Section
error	1	**CloseNicErr**	VipCloseNic() with invalid NicHandle *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER*	AS:9.2.2 / DG:3.1.2
error	1	**OpenNicErr**	VipOpenNic() with invalid DeviceName *case 1: Invalid string.* *case 2: NULL. ** *case 3: NULL string.* *Expect VIP_INVALID_PARAMETER*	AS:9.2.1 / DG:3.1.1
limit	1	**OpenNicLimit**	VipOpenNic() N+1 handles, then VipCloseNic(). *(A way to obtain the value of N is not in AS or IG, so the value must be configured by user)* *Expect VIP_SUCCESS and unique NicHandle until VIP_ERROR_RESOURCE; close one handle, then able to open another, VIP_SUCCESS while closing all.*	AS:9.2.1 + DG:3.1.1

hardware.xls

B.2. Endpoint Creation and Destruction

Type	TL	Name	Description	Section
basic	1	**CreateVi**	VipCreateVi() w/ each parameter value/combination, and ensure proper operation.	AS:9.3.1 / DG:3.2.1 AS:9.10.5 DG:4.6
			ViAttributes.ReliabilityLevel (per NicAttributes.ReliabilityLevelSupport):	*DG:4.5*
			VIP_SERVICE_UNRELIABLE	
			VIP_SERVICE_RELIABLE_DELIVERY	
			VIP_SERVICE_RELIABLE_RECEPTION	
			ViAttributes.MaxTransferSize:	
			(1 … NicAttributes.MaxTransferSize)	*AS:9.10.4 / DG:4.5*
			ViAttributes.QoS:	
			valid value	
			ViAttributes.Ptag:	
			(valid Ptag from VipCreatePtag)	
			ViAttributes.EnableRdmaWrite	
			TRUE, FALSE	
			ViAttributes.EnableRdmaRead	
			TRUE (if supported, per NicAttributes.RDMARead, for reliable connections only), FALSE	*DG:4.5*
	2		*SendCQHandle:*	
			NULL, Valid CQ	
	2		*RecvCQHandle:*	
			NULL, Valid CQ (same as SendCQHandle, different handle)	
			Expect VIP_SUCCESS for each	
	1		VipQueryVi() each of above	AS:9.8.3 / DG:3.7.3
			Expect VIP_SUCCESS, IDLE state, queues empty and same attributes	
	1		VipDestroyVi() each of above	AS:9.3.2 / DG:3.2.2
			Expect VIP_SUCCESS	
error	2	**CreateViAttribErr**	VipCreateVi() with invalid ViAttributes	AS:9.3.1 / DG:3.2.1
			case 1: Unsupported ViAttributes.ReliabilityLevel:	
			if applicable (per NicAttributes.ReliabilityLevelSupport), expect	*DG:4.5*
			VIP_INVALID_RELIABILITY_LEVEL	
			case 2: Invalid ViAttributes.QoS: (see 4.1 Assumptions)	
			expect VIP_INVALID_QOS	
			case 3: RDMA Read on Unreliable VI	*DG:4.5*
			case 4: RDMA Read if not supported (per NicAttributes.RDMARead),	*DG:4.5*
			expect VIP_INVALID_RDMAREAD	

error	1	**CreateViErr**	VipCreateVi() with invalid parameters *case 1: Previously destroyed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle (valid + 1). **	AS:9.3.1	/ DG:3.2.1
	2		*case 4: Previously destroyed SendCQHandle. **		
	2		*case 5: Previously destroyed RecvCQHandle. **		
	2		*case 6: Random SendCQHandle (valid + 1). **		
	2		*case 7: Random RecvCQHandle (valid + 1). **		
	2		*case 8: SendCQHandle registered on another NicHandle. **		
	2		*case 9: SendCQHandle registered on another NicHandle. **		
	2		*case 10: ViAttributes.Ptag registered with different NicHandle. **		
	2		*case 11: NULL ViAttributes.Ptag. **		
	2		*case 12: Random ViAttributes.Ptag. **		
			*case 13: Invalid ViAttributes.ReliabilityLevel. **	*AS:9.10.5*	*/ DG:4.5*
			*case 14: ViAttributes.MaxTransferSize > NicAttr.MaxTransferSize. **	*AS:9.10.4*	*/ DG:4.5*
	2		*case 15: Previously destroyed ViAttributes.Ptag. ** *Expect VIP_INVALID_PARAMETER (1-9), VIP_INVALID_PTAG (10-12, 15), VIP_INVALID_RELIABILITY_LEVEL (13) or VIP_INVALID_MTU (14).*		
limit	2	**CreateViLimit**	Attempt VipCreateVi() N+1 VIs *(N from NicAttributes.MaxVI attribute)* *Expect VIP_SUCCESS, VIP_ERROR_RESOURCE on N+1, then VIP_SUCCESS after closing a VI handle and creating another VI.*	AS:9.3.1 *AS:9.10.4*	/ DG:3.2.1 */ DG:4.5*
limit	2	**CreateViOv**	VipCreateVi() N VIs (until VIP_ERROR_RESOURCE), call VipCloseNic(), then repeat. *(N from NicAttributes.MaxVI attribute)* *Expect VIP_SUCCESS, VIP_ERROR_RESOURCE on N+1, then VIP_SUCCESS after closing NIC.*	AS:9.3.1 *AS:9.10.4*	/ DG:3.1.1 */ DG:4.5*
error	2	**DestroyViDescErr**	VipDestroyVi() with descriptors on work queue *case 1: Recv. Queue* *case 2: Send queue* *Expect VIP_ERROR_RESOURCE*	AS:9.3.2	/ DG:3.1.2
error	2	**DestroyViParamErr**	VipDestroyVi() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.3.2	/ DG:3.1.2

error	2	**DestroyViStateErr**	VipDestroyVi() with VI not in idle state *case 1: Pending Connect state* *case 2: Connected state* *case 3: Error state* *Expect VIP_ERROR_RESOURCE*	AS:9.3.2 / DG:3.1.2	

endpoint.xls

B.3. Connection Management

B.3.1. Client/Server

Type	TL	Name	Description	Section
error	1	**ConnAccept1**	VipConnectAccept() with mismatched MaxTransferSize (attribute of VI) then again w/ corrected MTS. *Expect VIP_INVALID_MTU then VIP_SUCCESS.*	AS:9.4.2 / DG:3.3.2
error	1	**ConnAccept2**	VipConnectAccept() with mismatched ReliabilityLevel (attribute of VI) then again w/ corrected Reliability. *Expect VIP_INVALID_RELIABILITY_LEVEL then VIP_SUCCESS.*	AS:9.4.2 / DG:3.3.2
error	1	**ConnAccept3**	VipConnectAccept() with invalid VI (already connected) then again w/ idle VI. *Expect VIP_INVALID_STATE then VIP_SUCCESS.*	AS:9.4.2 / DG:3.3.2
error	1	**ConnAccept4**	VipConnectAccept() with invalid VI (in error state) then again w/ idle VI. *Expect VIP_INVALID_STATE then VIP_SUCCESS.*	AS:9.4.2 / DG:3.3.2
error	2	**ConnAcceptCHanErr**	VipConnectAccept() with invalid ConnHandle. *case 1: NULL ConnHandle** *case 2: Previously accepted ConnHandle ** *case 3: Previously rejected ConnHandle ** *Expect VIP_INVALID_PARAMETER.*	AS:9.4.2 / DG:3.3.2
error	2	**ConnAcceptViHanErr**	VipConnectAccept() with invalid ViHandle. *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER.*	AS:9.4.2 / DG:3.3.2
basic	1	**ConnectAcceptTO**	VipConnectAccept() with stale ConnHandle. *(corresponding VipConnectRequest() has timed out)* *Expect VIP_TIMEOUT.*	AS:9.4.[2,4] + DG:3.3.[2,4]
basic	1	**ConnectReject**	VipConnectReject() on return from VipConnectWait().	AS:9.4.[1,3] / DG:3.3.[1,3]

			Expect VIP_SUCCESS on server, expect VIP_REJECT from client VipConnectRequest()	*AS:9.4.4*	/	*DG:3.3.4*
error	1	**ConnectRequestErr**	VipConnectRequest() with invalid parameters. *case 1: Previously destroyed ViHandle ** *case 2: NULL ViHandle** *case 3: Random ViHandle (valid + 1) ** *case 4: LocalAddr: NIC Address not correct* *case 5: LocalAddr: Discriminator Address too long* *case 6: Local Addr: NULL pointer ** *case 7: Timeout = 0* *case 8: RemoteAddr: NIC Address too short* *case 9: RemoteAddr: NIC Address too long* *case 10: RemoteAddr: Discriminator Address too long* *case 11: RemoteAddr: NULL pointer ** *Expect VIP_INVALID_PARAMETER for all.*	AS:9.4.4	/	DG:3.3.4
basic	1	**ConnectRequestINF**	Request valid VipConnectRequest() with VIP_INFINITE timeout. *wait a long time before accepting request . . .*	AS:9.4.4	/	DG:3.3.4
basic	1	**ConnectRequestTO**	Request valid VipConnectRequest() that is never satisfied. *(requires VipConnectWait() outstanding on server, or VIP_NO_MATCH will be returned)* *verify timeout value is honored, VIP_TIMEOUT returned.*	AS:9.4.4	/	DG:3.3.4
error	1	**ConnectWaitErr**	VipConnectWait() with invalid parameters. *case 1: Previously closed NicHandle ** *case 2: NULL NicHandle** *case 3: Random NicHandle (valid + 1) ** *case 4: LocalAddr: NIC Address not correct* *case 5: LocalAddr: Discriminator Address too long* *case 6: Local Addr: NULL pointer ** *case 7: Timeout = 0* *Expect VIP_INVALID_PARAMETER for cases 1-6, VIP_TIMEOUT for case 7.*	AS:9.4.1	/	DG:3.3.1
basic	1	**ConnectWaitTO**	Request valid VipConnectWait() that is never satisfied. *verify timeout value is honored, VIP_TIMEOUT returned.*	AS:9.4.1	/	DG:3.3.1
limit	2	**ConnOv**	Make N connections, then call VipCloseNic() and retry. *(N from VipQueryNic() MaxVI attribute)* *Expect VIP_SUCCESS until VIP_ERROR_RESOURCE, able to create more after closing/reopening NIC.*	AS:9.4.4	/	DG:3.3.4
error	2	**ConnRejectCHanErr**	VipConnectReject() with invalid ConnHandle.	AS:9.4.3	/	DG:3.3.3

			*case 1: NULL ConnHandle** *case 2: Previously accepted ConnHandle ** *case 3: Previously rejected ConnHandle ** *Expect VIP_INVALID_PARAMETER.*			
stress	1	**CSconn**	Multi-threaded test with multiple simultaneous c/s connection requests. *Expect VIP_SUCCESS for each connection.*	AS:9.4.4	/	DG:3.3.4
basic	1	**CSDiscriminator**	Valid c/s connection establishment w/ all supported discriminator lengths *VI state VIP_STATE_CONNECTED.* *(discriminator length 0..NIC MaxDiscriminatorLen attribute)* *Verify expected RemoteAddr, RemoteViAttribs supplied, VIP_SUCCESS on all, and*	AS:9.4.[1,4] *AS:9.10.4*	/ /	DG:3.3.[1,4] *DG:4.5*
error	2	**DisconnectErr**	VipDisconnect() with invalid ViHandle. *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER.*	AS:9.4.5	/	DG:3.3.5

connect.c-s.xls

B.3.2. Peer to Peer

Type	TL	Name	Description	Section
basic	2	**ConnectPeerCS**	VipConnectPeerRequest() does not match VipConnectRequest() request. *Expect both sides should time out, return VIP_TIMEOUT.*	DG:3.3.[1,6]
basic	2	**ConnectPeerDoneTO**	Valid VipConnectPeerRequest() that is never satisfied. *(various timeout values)* *verify timeout value is honored, VIP_NOT_DONE then VIP_TIMEOUT returned after timeout.*	DG:3.3.[6,7]
		ConnectPeerWaitTO	*verify timeout value is honored, then VIP_TIMEOUT returned after timeout.*	
error	2	**ConnPeerDoneErr**	VipConnectPeerDone() with invalid ViHandle. *case 1: Previously destroyed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle (valid + 16). ** *case 4: Vi in idle state.* *Expect VIP_INVALID_PARAMETER (cases 1-3) or VIP_INVALID_STATE (case 4).*	DG:3.3.7
error	2	**ConnPeerReqAttribErr**	VipConnectPeerRequest() with mismatched VI attribute. *case 1: VipConnectPeerWait with mismatched ReliabilityLevel.* *case 2: VipConnectPeerDone with mismatched ReliabilityLevel.* *case 3: VipConnectPeerWait with mismatched MTU.* *case 4: VipConnectPeerDone with mismatched MTU.* *Expect VIP_INVALID_PARAMETER.*	DG:3.3.6
error	2	**ConnPeerReqErr**	VipConnectPeerRequest() with invalid parameters. *case 1: Previously destroyed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle (valid + 16). ** *case 4: LocalAddr: NIC Address not correct.* *case 5: LocalAddr: Discriminator Address too long.* *case 6: Local Addr: NULL pointer. ** *case 7: Timeout = 0.* *case 8: RemoteAddr: NIC Address too short.* *case 9: RemoteAddr: NIC Address too long.* *case 10: RemoteAddr: Discriminator Address too long.* *case 11: RemoteAddr: NULL pointer. ** *Expect VIP_INVALID_PARAMETER.*	DG:3.3.6
basic	2	**ConnectPeerSelf**	VipConnectPeer*() connections to self.	DG:3.3.[6,8]

			(Multi-threaded test) *Expect VIP_SUCCESS for all VipConnectPeer*() calls.*		
error	2	**ConnPeerWaitErr**	VipConnectPeerWait() with invalid ViHandle. *case 1: Previously destroyed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle (valid + 16). ** *case 4: Vi in idle state.* *Expect VIP_INVALID_PARAMETER (cases 1-3) or VIP_INVALID_STATE (case 4).*		DG:3.3.8
basic	2	**Disconnect**	VipDisconnect() on VI with peer connection in progress. *Expect VIP_SUCCESS, VI state is VIP_STATE_CONNECT_PENDING before disconnect, VIP_STATE_IDLE afterwards).*	AS:9.4.6	DG:3.3.[6-8]
stress	2	**PPconn**	Multi-threaded test with multiple simultaneous p/p connection requests. *Expect VIP_SUCCESS for each connection.*		DG:3.3.6
basic	2	**PPDiscriminator**	Valid p/p connection establishment w/ all supported discriminator lengths *(discriminator length 0..NIC MaxDiscriminatorLen attribute)* *Verify expected RemoteViAttribs supplied, VIP_SUCCESS on all, and VI state VIP_STATE_CONNECTED.*		DG:3.3.6 *DG:4.5*

connect.peer.xls

B.4. Memory Protection and Registration

Type	TL	Name	Description	Section
error	2	**CreatePtagErr**	VipCreatePtag() with invalid NicHandle. *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle (valid + 1). ** *Expect VIP_INVALID_PARAMETER.*	AS:9.5.1 / DG:3.4.1
limit	2	**CreatePtagLimit**	VipCreatePtag() N+1 tags, then VipDestroyPtag() *(N from VipQueryNic() MaxPtags attribute)* *Expect VIP_SUCCESS until VIP_ERROR_RESOURCE, destroy one, then able to open more after destroying.*	AS:9.5.1 / DG:3.4.1 *AS:9.10.4 / DG:4.5*
limit	2	**CreatePtagOv**	VipCreatePtag() N tags (until VIP_ERROR_RESOURCE), call VipCloseNic(), then repeat. *(N from VipQueryNic() MaxPtags attribute)* *Expect VIP_SUCCESS (I.e. Ptags cleaned up)*	AS:9.5.1 / DG:3.4.1 *AS:9.10.4 / DG:4.5*
error	1	**DeregisterMemErr**	VipDeregisterMem() with invalid parameters. *case 1: Previously destroyed MemHandle. ** *case 2: NULL MemHandle. ** *case 3: Random MemHandle. ** *case 4: Different NicHandle than register. ** *case 5: NULL NicHandle. ** *case 6: Random NicHandle (valid + 1). ** *case 7: Different VirtualAddress (0). ** *Expect VIP_INVALID_PARAMETER.*	AS:9.5.4 / DG:3.4.4
error	2	**DestroyPtagErr**	VipDestroyPtag() with invalid NicHandle. *case 1: Previously destroyed MemHandle. ** *case 2: NULL MemHandle. ** *case 3: Random MemHandle ** *case 4: Different NicHandle than register. ** *case 5: NULL NicHandle. ** *case 6: Random NicHandle (valid + 1). ** *Expect VIP_INVALID_PARAMETER.*	AS:9.5.2 / DG:3.4.2
error	2	**DestroyPtagUseErr**	VipDestroyPtag() with ProtectionTag still in use.	AS:9.5.2 / DG:3.4.2

			case 1: VI *case 2: Memory Region* *Expect VIP_ERROR_RESOURCE.*			
basic	1	**RegisterMem**	VipRegisterMem() w/ each parameter value/combination, and ensure proper operation, then VipDeregisterMem(). *VirtualAddress* *(start on page boundary, …+1 byte, …+2 bytes)* *Length* *(1 .. MaxRegisterBytes from VipQueryNic())* *MemAttributes.EnableRdmaWrite* *TRUE,FALSE* *MemAttributes.EnableRdmaRead* *TRUE (only if allowed), FALSE* *(based on NicAttribute.RDMARead)* *Expect VIP_SUCCESS.*	AS:9.5.3 *AS:9.10.4* *AS:9.10.6* *AS:9.10.6* 	/ / / / 	DG:3.4.3 *DG:4.5* *DG:4.7* *DG:4.7* *DG:4.5*
	3		VipQueryMem() w/ each of above *Expect VIP_SUCCESS and correct MemAttributes.*	AS:9.8.5	/	DG:3.7.5
error	1	**RegisterMemErr**	VipRegisterMem() with invalid parameters. *case 1: Closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *case 4: Invalid Virtual Address (address 0).* *case 5: Previously destroyed Ptag (MemAttr.Ptag).* *case 6: NULL Ptag (MemAttr.Ptag).* *case 7: Random Ptag (valid + 16) (MemAttr.Ptag)* *case 8: Ptag registered to another NicHandle.* *case 9: NULL MemHandle.* *case 10: Random MemHandle.* *Expect VIP_ERROR_PARAMETER (cases 1-6) or VIP_INVALID_PTAG (cases 7-10), returned MemHandle == NULL.*	AS:9.5.3	/	DG:3.4.3
	2					
	2					
	2					
	2					
limit	2	**RegisterMemLimit**	VipRegisterMem() N+1 regions, then VipDeregisterMem() *(N from VipQueryNic() MaxRegisterRegions attribute)* *Expect VIP_SUCCESS until VIP_ERROR_RESOURCE, deregister one, then able to register more after deregistering.*	AS:9.5.3 *AS:9.10.4*	/ /	DG:3.4.1 *DG:4.5*
basic	2	**RegisterMemOverlap**	VipRegisterMem() overlapping region, VipDeregisterMem() one, then ensure remaining registered region is still usable. *Expect VIP_SUCCESS.*	AS:9.5.3	+	DG:3.4.3

limit	2	**RegisterMemOv**	VipRegisterMem() N regions (until VIP_ERROR_RESOURCE), call VipCloseNic(), then repeat. *(N from VipQueryNic() MaxRegisterRegions attribute)* *Expect VIP_SUCCESS until VIP_ERROR_RESOURCE, close NIC, then able to register more.*	AS:9.5.3 *AS:9.10.4*	/	DG:3.4.3 *DG:4.5*
error	3	**SetMemAttrErr**	VipRegisterMem() with invalid parameters *case 1: Closed NicHandle.* *case 2: NULL NicHandle.* *case 3: Random NicHandle (valid + 16).* *case 4: Invalid VirtualAddress (0).* *case 5: Previously destroyed Ptag (MemAttr.Ptag).* *case 6: NULL Ptag (MemAttr.Ptag).* *case 7: Random Ptag (valid + 16) (MemAttr.Ptag)* *case 8: Ptag registered to another NicHandle.* *case 9: NULL MemHandle.* *case 10: Random MemHandle.* *Expect VIP_ERROR_PARAMETER (cases 1-4,9,10) or VIP_INVALID_PTAG (cases 5-8)*	AS:9.5.3	/	DG:3.4.3

memory.xls

B.5. Data Transfer and Completion Operations

Type	TL	Name	Description	Section
error	2	**CQDoneHanErr**	VipCQDone() with invalid CQHandle *case 1: NULL CQHandle ** *case 2: Previously destroyed CQHandle ** *case 3: Invalid CQHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.7 DG:3.5.7
error	3	**CQNotifyHanErr**	VipCQNotify() with invalid ViHandle *case 1: NULL CQHandle ** *case 2: Previously destroyed CQHandle ** *case 3: Invalid CQHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.11 DG:3.5.11
basic	2	**CQreverse**	Test descriptor completion semantics with completion queues. *Call VipSendDone()/VipRecvDone() before VipCQDone()/VipCQWait().* *Expect success.*	AS:6.4.1
error	2	**CQWaitHanErr**	VipCQWait() with invalid CQHandle *case 1: NULL CQHandle ** *case 2: Previously destroyed CQHandle ** *case 3: Invalid CQHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.8 DG:3.5.8
basic	2	**CQWaitTO**	Request valid VipCQDone() with no descriptors on queue. *Expect VIP_NOT_DONE returned.*	AS:9.6.7 / DG:3.5.7
			Request valid VipCQWait() (I.e. descriptor on queue) that is never satisfied *(0, 1 second, 10 seconds, 2 minutes)* *Expect timeout value is honored, VIP_TIMEOUT returned.*	AS:9.6.8 / DG:3.5.8
stress	2	**MultiCQ1**	Multi-threaded sends/receives on a single VI with completion queue, using *Done semantics. *Requires thread-safe implementation.*	AS:9.6 / DG:3.5
stress	2	**MultiCQ2**	Multi-threaded sends/receives on a single VI with completion queue, using *Wait semantics. *Requires thread-safe implementation.*	AS:9.6 / DG:3.5

stress	2	**MultiCQ3**	Multi-threaded sends/receives on a single VI with completion queue, using mixed *Done/*Wait semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	3	**MultiCQ4**	Multi-threaded sends/receives on a single VI with completion queue, using *Notify semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	1	**MultiThread1**	Multi-threaded sends/receives on a single VI, using *Done semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	1	**MultiThread2**	Multi-threaded sends/receives on a single VI, using *Wait semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	1	**MultiThread3**	Multi-threaded sends/receives on a single VI, using mixed *Done/*Wait semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	3	**MultiThread4**	Multi-threaded sends/receives on a single VI, using *Notify semantics. *Requires thread-safe implementation.*	AS:9.6	/ DG:3.5
stress	1	**MultiVISend**	Send/receives on multiple simultaneous VIs (multi-threaded test).	AS:9.6	/ DG:3.5
error	1	**PostRecvErr**	VipPostRecv() with invalid ViHandle *case 1: Closed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle. ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.4	/ DG:3.5.4
error	1	**PostSendErr**	VipPostSend() with invalid ViHandle *case 1: Closed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle. ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.1	/ DG:3.5.1
basic	1	**RdmaWrite**	Exercise RDMA Write basic functionality on one VI *Between each NIC, ping-pong. Corresponding receives are posted before VI connected and each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:*	AS:9.6 *AS:9.6.[1,4]*	/ DG:3.5 */ DG:3.5.[1,4]*

			0..MaxTransferSize *Immediate Data:* *with/without* *Completion semantics (send side, recv side w/ immediate data):* *VipSendWait(),VipRecvWait()* *Other attributes:* *odd/even memory addresses* *single data descriptors* *Expect success for all, data correct, descriptors correct, no other areas in buffers modified.*	*AS:9.6.[3,6] / DG:3.5.[3,6]*
basic	3	**RecvCQNotify**	Test Completion Queue Notify basic functionality w/ CQ attached to Recv Queue. *Same as SendRecv, except VipCQNotify is called using a CQ that has been attached to the recv queue instead of VIpRecvWait().*	AS:9.6.11 / DG:3.5.11
error	2	**RecvDoneViHanErr**	VipRecvDone() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.5 DG:3.5.5
basic	1	**RecvDS**	Test Send/Recv functionality on one VI with multiple recv segments per descriptor. *Between each NIC, ping-pong. Corresponding receives are posted before VI is connected and before each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *MaxTransferSize* *Immediate Data:* *with/without* *Completion semantics:* *VipSendWait(),VipRecvWait()* *Other attributes:* *2..MaxSegmentsPerDesc receive segments/descriptor, single send segment* *Expect success for all, data correct, descriptors correct*	AS:9.6 / DG:3.5 *AS:9.6.[1,4] / DG:3.5.[1,4]* *AS:9.6.[3,6] / DG:3.5.[3,6]*
basic	1	**RecvDSzero**	Test Send/Recv functionality on one VI with multiple 0-length recv segments per descriptor. *Between each NIC, ping-pong. Corresponding receives are posted before VI is connected and before each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *0/1 length segments of various patterns, total 0..MaxSegmentsPerDesc*	AS:9.6 / DG:3.5 *AS:9.6.[1,4] / DG:3.5.[1,4]*

			Completion semantics: *VipSendDone(),VipRecvDone()* *Other attributes:* *MaxSegmentsPerDesc receive segments/descriptors, single send segment* *Expect success for all, data correct, descriptors correct.*	*AS:9.6.[2,5]*	/ *DG:3.5.[2,5]*
basic	1	**RecvImmed**	VipPostRecv() with Immediate Data set in Control Segment *(should be ignored)* *Expect VIP_SUCCESS*	AS:9.6.4 *AS:10.2*	/ DG:3.5.4 *+ DG:5.2*
basic	1	**RecvNone**	Request VipRecvDone() with no descriptors on queue. *Expect NULL DescriptorPtr and VIP_DESCRIPTOR_ERROR returned.*		G:3.5.5
			Request VipRecvWait() with no descriptors on queue. *Expect timeout value is not honored, NULL DescriptorPtr and VIP_DESCRIPTOR_ERROR returned.*		DG:3.5.6
basic	3	**RecvNotify**	Test Recv Notify basic functionality on one VI *Same as SendRecv, except VipRecvNotify() will be used instead of VipRecvWait().*	AS:9.6.10	/ DG:3.5.10
error	3	**RecvNotifyCQErr**	VipRecvNotify() on VI associated with Completion Queue *Expect VIP_ERROR_RESOURCE*	AS:9.6.11	DG:3.5.10
error	3	**RecvNotifyViHanErr**	VipRecvNotify() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.10	DG:3.5.10
error	2	**RecvWaitCQErr**	VipRecvWait() on VI associated with Completion Queue *Expect VIP_ERROR_RESOURCE*	AS:9.6.6	DG:3.5.6
basic	1	**RecvWaitTO**	Request valid VipRecvDone() with descriptors on queue not completed. *Expect VIP_NOT_DONE returned.*	AS:9.6.5	/ DG:3.5.5
			Request valid VipRecvWait() (I.e. descriptor on queue) that is never satisfied *(0, 1 second, 10 seconds, 2 minutes)* *Expect timeout value is honored, VIP_TIMEOUT returned.*	AS:9.6.6	/ DG:3.5.6
error	2	**RecvWaitViHanErr**	VipRecvWait() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle **	AS:9.6.6	DG:3.5.6

			*case 3: Invalid ViHandle (valid+2*PageSize) * Expect VIP_INVALID_PARAMETER*		
basic	3	**SendCQNotify**	Test Completion Queue Notify basic functionality w/ CQ attached to Send Queue. *Same as SendRecv, except VipCQNotify is called using a CQ that has been attached to the send queue instead of VIpSendWait().*	AS:9.6.11	/ DG:3.5.11
error	2	**SendDoneViHanErr**	VipSendDone() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.2	DG:3.5.2
basic	1	**SendDS**	Test Send/Recv functionality on one VI with multiple send segments per descriptor. *Between each NIC, ping-pong. Corresponding receives are posted before VI is connected and before each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *MaxTransferSize* *Immediate Data:* *with/without* *Completion semantics:* *VipSendDone(),VipRecvDone()* *Other attributes:* *2..MaxSegmentsPerDesc send segments/descriptors, single receive segment* *Expect success for all, data correct, descriptors correct.*	AS:9.6 *AS:9.6.[1,4]* *AS:9.6.[2,5]*	/ DG:3.5 */ DG:3.5.[1,4]* */ DG:3.5.[2,5]*
basic	1	**SendDSzero**	Test Send/Recv functionality on one VI with multiple 0-length send segments per descriptor. *Between each NIC, ping-pong. Corresponding receives are posted before VI is connected and before each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *0/1 length segments of various patterns, total 0..MaxSegmentsPerDesc* *Completion semantics:* *VipSendWait(),VipRecvWait()* *Other attributes:* *MaxSegmentsPerDesc send segments/descriptors, single receive segment* *Expect success for all, data correct, descriptors correct.*	AS:9.6 *AS:9.6.[1,4]* *AS:9.6.[3,6]*	/ DG:3.5 */ DG:3.5.[1,4]* */ DG:3.5.[3,6]*
basic	1	**SendNone**	Request VipSendDone() with no descriptors on queue.		DG:3.5.2

			Expect NULL DescriptorPtr and VIP_DESCRIPTOR_ERROR returned.		
			Request VipSendWait() with no descriptors on queue. *(0, 10 seconds, VIP_INFINITE)* *Expect timeout value is not honored, NULL DescriptorPtr and VIP_DESCRIPTOR_ERROR returned.*		DG:3.5.3
basic	3	**SendNotify**	Test Send Notify basic functionality on one VI *Same as SendRecv, except VipSendNotify() will be used instead of VipSendWait().*	AS:9.6.9	/ DG:3.5.9
error	3	**SendNotifyCQErr**	VipSendNotify() on VI associated with Completion Queue *Expect VIP_ERROR_RESOURCE*	AS:9.6.AS:9	DG:3.5.9
error	3	**SendNotifyViHanErr**	VipSendNotify() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.AS:9	DG:3.5.9
basic	1	**SendRecv**	Test Send/Recv basic functionality on one VI *Between each NIC, ping-pong. Corresponding receives are posted before VI is connected and before each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *0..MaxTransferSize* *Immediate Data:* *with/without* *Completion semantics:* *VipSendWait(),VipRecvWait()* *Other attributes:* *odd/even memory addresses for buffers* *single data descriptors* *Expect success for all, data correct, descriptors correct.*	AS:9.6 *AS:9.6.[1,4]* *AS:9.6.[3,6]* *AS:10*	/ DG:3.5 */ DG:3.5.[1,4]* */ DG:3.5.[3,6]* */ DG:5*
basic	2	**SendRecvCQ**	Test completion queue semantics (1 CQ for both send/recv) *otherwise identical to SendRecv test.*	AS:9.6.8	/ DG:3.5.8
basic	1	**SendSelf**	Same as SendRecv, but loopback on a NIC *multi-threaded test*	AS:9.6	/ DG:3.5

error	2	**SendWaitViHanErr**	VipSendWait() with invalid ViHandle *case 1: NULL ViHandle ** *case 2: Previously destroyed ViHandle ** *case 3: Invalid ViHandle (valid+2*PageSize) ** *Expect VIP_INVALID_PARAMETER*	AS:9.6.3	DG:3.5.3
basic	1	**WriteDS**	RDMA Write with multiple data segments on one VI. *Between each NIC, ping-pong. Corresponding receives are posted before VI connected and each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *MaxTransferSize* *Immediate Data:* *with/without* *Completion semantics (send side, recv side w/ immediate data):* *VipSendWait(),VipRecvWait()* *Other attributes:* *2..MaxSegmentsPerDesc send segments.* *Expect success for all, data correct, descriptors correct, no other areas in buffers modified.*	AS:9.6 / *AS:9.6.[1,4]* *AS:9.6.[3,6]*	DG:3.5 / *DG:3.5.[1,4]* *DG:3.5.[3,6]*
basic	1	**WriteDSzero**	RDMA Write with multiple 0-length data segments per send descriptor. *Between each NIC, ping-pong. Corresponding receives are posted before VI connected and each send.* *Reliability:* *Unreliable, Reliable Delivery, Reliable Reception* *Length:* *0/1 length segments of various patterns, total 0..MaxSegmentsPerDesc* *Other attributes:* *MaxSegmentsPerDesc send segments/descriptor* *Expect success for all, data correct, descriptors correct, no other areas in buffers modified*	AS:9.6 / *AS:9.6.[1,4]*	DG:3.5 / *DG:3.5.[1,4]*
basic	1	**WriteSelf**	Same as RdmaWrite, but loopback on a NIC *multi-threaded test*	AS:9.6	/ DG:3.5

data.xls

B.6. Completion Queue Management

Type	TL	Name	Description	Section
basic	2	**CreateCQ**	Create, resize and destroy a completion queue. *EntryCount:* *1 .. NicAttributes.MaxCQEntries* *Expect VIP_SUCCESS.*	AS:9.7 / DG:3.6 *AS:9.10.4 / DG:4.5*
error	2	**CreateCQErr**	VipCreateCQ() with invalid parameters. *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER.* *case 4: Invalid EntryCount (=0).* *case 5: Invalid EntryCount (> NicAttr.MaxCQEntries)* *Expect VIP_ERROR_RESOURCE.*	AS:9.7.1 / DG:3.6.1 *AS:9.10.4 / DG:4.5*
limit	2	**CreateCQlimit**	Attempt N+1 queues, then destroy queue. *(N from NicAttributes.MaxCQ)* *Expect VIP_SUCCESS until VIP_ERROR_RESOURCE, able to create more after destroying.*	AS:9.7.1 / DG:3.6.1 *AS:9.10.4 / DG:4.5*
limit	2	**CreateCQOv**	Attempt N+1 queues, then call VipCloseNic() and retry. *(N from NicAttributes.MaxCQ)*	AS:9.7.1 / DG:3.6.1
error	2	**DestroyCQErr**	VipDestroyCQ() with invalid parameters. *case 1: Previously destroyed CQHandle. ** *case 2: NULL CQHandle. ** *case 3: Random CQHandle. ** *Expect VIP_INVALID_PARAMETER.* *case 4: with CQHandle associated to VI work queue.* *Expect VIP_ERROR_RESOPURCE.*	AS:9.7.2 / DG:3.6.2
stress	3	**ResizeCQ**	Test completion queue resize semantics on active VI. (multi-threaded test) *Similar to MultiCQ tests, with added VipResize CQ calls.*	AS:9.7.3 / DG:3.6.3

error	2	**ResizeCQErr**	VipResizeCQ() with invalid parameters. *case 1: Previously destroyed CQHandle. ** *case 2: NULL CQHandle. ** *case 3: Random CQHandle. ** *Expect VIP_INVALID_PARAMETER.* *case 4: Invalid EntryCount (=0).* *case 5: Invalid EntryCount (> NicAttr.MaxCQEntries)* *Expect VIP_ERROR_RESOURCE.*	AS:9.7.3 *AS:9.10.4*	/ DG:3.6.3 */ DG:4.5*

NOTE: Correct operation of Completion Queues is described with Data Transfer and Completion Operations

cq.xls

B.7. Querying Information

Type	TL	Name	Description	Section
error	3	**QueryMemErr**	VipQueryMem() w/ invalid parameters. *case 1: Different NicHandle than register. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle (valid + 1). ** *case 4: Different VirtualAddress (0). ** *case 5: Previously destroyed MemHandle. ** *case 6: NULL MemHandle. ** *case 7: Random MemHandle. ** *Expect VIP_INVALID_PARAMETER.*	AS:9.8.5 / DG:3.7.5
basic	1	**QueryNic**	VipQueryNic() functional test. *MaxMTU >= 32K* *MaxSegmentsPerDesc >= 252* *MaxCQEntries >= 1024* *MaxDiscriminatorLen >= 16* *At least one Ptag per supported VI* *Expect VIP_SUCCESS, NIC attributes data structure is completed correctly, with the above restrictions met.*	AS:9.8.1 / DG:3.7.1 *AS:6.1.1.2* *AS:6.1.1.1 / DG:5.4* *AS:3.4* *DG:4.5* *AS:9.10.4 / DG:4.5*
error	1	**QueryNicErr**	VipQueryNic() with invalid NIC handle. *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER.*	AS:9.8.1 / DG:2.7.1
basic	3	**QuerySMInfo**	VipQuerySystemManagementInfo() with InfoType=VIP_SMI_AUTODISCOVERY. *Expect VIP_SUCCESS, VIP_AUTODISCOVERY_LIST returned.* *(NOTE: The contents of the VIP_AUTODISCOVERY_LIST will need to be manually verified).*	AS:9.8.6 + DG:3.7.6

error	3	**QuerySMInfoErr**	VipQuerySystemManagementInfo() with invalid parameter. *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER*	AS:9.8.6 *AS:9.8.6*	+ /	DG:3.7.6 *DG:3.7.6* *DG:3.7.6*
error	2	**QueryViErr**	VipQueryVi() with invalid parameters. *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER.*	AS:9.8.3	/	DG:3.7.3
basic	3	**SetMemAttr**	VipSetMemAttributes() w/ each parameter value, and ensure proper operation. *Address (same as RegisterMem() value)* *MemHandle (same as RegisterMem() value)* *MemAttribs:* *.PTAG (alternate between 2 valid Ptags)* *.EnableRDMARead (TRUE (iff NicAttr.RDMARead is TRUE),* *FALSE)* *.EnableRDMAWrite (TRUE, FALSE)* *Expect VIP_SUCCESS for all.*	AS:9.8.4 *AS:9.10.6* *AS:9.10.6*	/ / /	DG:3.7.4 *DG:4.7* *DG:4.7* *DG:4.5* *DG:4.7*
			VipQueryMem() w/ each of above *Expect VIP_SUCCESS, and correct MemAttribs.*	AS:9.8.5	/	DG:3.7.5
error	3	**SetMemAttrErr**	VipSetMemAttributes() with invalid parameters. *case 1: Closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *case 4: Invalid Virtual Address (address 0).* *case 5: Length = 0.* *case 6 Destroyed Ptag. ** *case 7: NULL Ptag. ** *case 8: Random Ptag. ** *case 9: Ptag registered to another NicHandle. **	AS:9.8.4	/	DG:3.7.4

			*case 10: Destroyed MemHandle. ** *case 11: NULL MemHandle. ** *case 12: Random MemHandle. ** *Expect VIP_ERROR_PARAMETER (cases 1-5,10-12) or VIP_INVALID_PTAG (cases 6-9), returned MemHandle == NULL, attributes are not changed.*		
error	3	**SetViAttrErr**	VipSetViAttributes() w/ invalid parameters. *case 1: Previously destroyed ViHandle. ** *case 2: NULL ViHandle. ** *case 3: Random ViHandle (valid + 16). ** *case 4: ViAttributes.Ptag registered with different NicHandle. ** *case 5: NULL ViAttributes.Ptag. ** *case 6: Random ViAttributes.Ptag. ** *case 7: Invalid ViAttributes.ReliabilityLevel.* *case 8: ViAttributes.MaxTransferSize > NicAttr.MaxTransferSize* *case 9: Previously destroyed ViAttributes.Ptag. ** *Expect VIP_INVALID_PARAMETER (cases 1-3), VIP_INVALID_PTAG (cases 4-6, 9), VIP_INVALID_RELIABILITY_LEVEL (case 7) or VIP_INVALID_MTU (case 8).*	AS:9.8.4	+ DG:3.7.4
basic	3	**SetViAttrib**	VipSetViAttributes() w/ each parameter value on an idle VI. *ViAttributes.ReliabilityLevel:* *VIP_SERVICE_UNRELIABLE* *VIP_SERVICE_RELIABLE_DELIVERY* *VIP_SERVICE_RELIABLE_RECEPTION* *ViAttributes.MaxTransferSize:* *(1 … NicAttributes.MaxTransferSize)* *ViAttributes.QoS:* *values (see 4.1 Assumptions)* *ViAttributes.Ptag:* *(valid Ptags from VipCreatePtag)* *ViAttributes.EnableRdmaWrite* *FALSE, TRUE* *ViAttributes.EnableRdmaRead* *FALSE, TRUE (for Reliable connections only, iff NicAttr.RDMARead is TRUE)* *Expect VIP_SUCCESS.*	AS:9.8.2 *AS:9.10.5* *AS:9.10.5* *AS:9.10.4* *AS:9.10.5* *AS:9.10.5* *AS:9.5.1* *AS:9.10.5* *AS:9.10.5* 	+ DG:3.7.2 */ DG:4.6* */ DG:4.6* */ DG:4.5* */ DG:4.6* */ DG:4.6* */ DG:3.4.1* */ DG:4.6* */ DG:4.6*

			VipQueryVi() each of above. *Expect VIP_SUCCESS, IDLE state and correct attributes of the VI.*	AS:9.8.3	/	DG:3.7.3
error	3	**SetViAttrInvErr**	VipSetViAttributes() with invalid ViAttributes. *ViAttributes.QoS (see 4.1 Assumptions):* *expect VIP_INVALID_QOS* *ViAttributes.EnableRdmaRead(TRUE, on unreliable only):* *expect VIP_INVALID_RDMAREAD* *Proper error returned, and other attributes were not changed.*	AS:9.8.2 *AS:9.10.5* *AS:9.10.5* 	+ */* */* 	DG:3.7.2 *DG:4.6* *DG:4.6* *DG:3.7.2*
error	3	**SetViAttrStateErr**	VipSetViAttributes() call with VI in invalid (non-idle) state. *VIP_STATE_CONNECTED* *VIP_STATE_CONNECT_PENDING* *VIP_STATE_ERROR* *Expect VIP_INVALID_STATE.*	AS:9.8.2	+	DG:3.7.2 *DG:3.7.2*

query.xls

B.8. Error Handling

```
Naming convention for Async. Error tests:
ErrH    Snd     Form        1
        Rcv     Ov          2
        Rdw     Len         3
        Rdwi    Post
        Rdr     Prot
                CompProt
                RcvQEmpty
```

Type	TL	Name	Description	Section
async	2	**ErrHCQ1**	VipPostSend() with Error in Desc using CQ (send/recv on same CQ) *Post 3 recv descriptors. Then post 3 send descriptors, with the first having a descriptor* format error. *Sender: Expect VipCQDone() to specify Send Queue and send descriptor with VIP_STATUS_FORMAT_ERROR in the status plus:* Unreliable Delivery: *VIs remain connected, all further sends/recvs complete successfully.* Reliable Delivery/Reception: *Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, completion queue entries for all of the outstanding request descriptors, outstanding descriptors on both work queues completed in error.* *Receiver: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.*	AS:9.6.7 / DG:3.5.7 *AS:2.5.1* *AS:2.5.[2,3]* + *DG:6.2*
async	2	**ErrHCQ2**	VipPostSend() with Error in Desc using CQ (send/recv on different CQs) *Post 3 recv descriptors. Then post 3 send descriptors, with the first having a descriptor* format error. *Sender: Expect VipCQDone() to specify Send Queue and send descriptor with VIP_STATUS_FORMAT_ERROR in the status plus:* Unreliable Delivery: *VIs remain connected, all further sends/recvs complete successfully.* Reliable Delivery/Reception: *Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, completion queue entries for all of the outstanding request descriptors, outstanding descriptors on both work queues completed in error.*	AS: 9.6.7 / DG:3.5.7 *AS:2.5.1* *AS:2.5.[2,3]* *DG:6.2*

Receiver: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.

async	2	**ErrHDefault**	Default handler is returned by VipErrorCallback() with Handler = NULL *Register error handler with VipErrorCallback(). Then call VipErrorCallback() with a NULL handler. Post a send descriptor with no corresponding recv descriptor, which should result in an async callback (VIP_ERROR_RECVQ_EMPTY) on the receive side and error callback (VIP_ERROR_CONN_LOST) on the send side. Expect:* *VipErrorCallback should return VIP_SUCCESS plus* Unreliable delivery: *Expect no notification on either side, VIs remain connected* Reliable Delivery: Sender: error callback function is never invoked, Default error handler is invoked with *VIP_ERROR_CONN_LOST, VI in error state.* *Receiver: Expect error callback function is never invoked, Default error handler is invoked with VIP_ERROR_RECVQ_EMPTY.* *RECOMMENDED: Default error handler is invoked with VIP_ERROR_CONN_LOST. VI in error state.* Reliable Reception: *Sender: expect VipSendWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_REMOTE_DESC_ERROR in the status. Error callback function is never invoked.* *RECOMMENDED: Default error handler is invoked with VIP_ERROR_CONN_LOST. VI in error state, queues flushed* *Receiver: same as Reliable Delivery.* *Note: Since the behavior of the default error handler is not completely specified, it's behavior in this test must be verified manually.*	AS:9.1 *AS:2.5.1* *AS:2.5.2* *AS:2.5.3*	/ DG:3.8.1
async	2	**ErrHRcvComp**	VipPostSend() Send Operation with invalid Completion Queue Reliable Delivery: *Sender: expect send and disconnect to complete successfully. All outstanding request descriptors completed in error on both queues.* RecvDone should return VIP_DESCRIPTOR_ERROR w/VIP_STATUS_DONE and *and VIP_STATUS_DESC_FLUSHED_ERROR in the status field* or VIP_DESCRIPTOR_ERROR with a NULL descriptor. *RECOMMENDED: Errorhandler is invoked with VIP_ERROR_CONN_LOST.* *Receiver:* *expect error callback (VIP_ERROR_CONN_LOST), VIP_DESCRIPTOR_ERROR w/* VIP_STATUS_DONE and VIP_STATUS_DESC_FLUSHED_ERROR in the status.	AS:2.5 *AS:2.5.2*	/ DG:3.8

			Reliable Reception: *Sender: same as Reliable Delivery* *Receiver:* *VipPostSend() should return VIP_SUCCESS.* *VipSendDone() returns VIP_ERROR_DESCRIPTOR_ERROR,* *status=REMOTE_DESC_ERROR* *RECOMMENDED: ErrorHandler is invoked with VIP_ERROR_CONN_LOST.*	*AS:2.5.3*	
async	2	**ErrHRcvForm**	VipPostRecv() with Descriptor Format Errors. Case 1: Operation = 3 (invalid operation) Case 2: Control Segment Reserved != 0 Case 3: Control Field bits 15:4 != 0 Case 4: Operation = CONTROL_OP_RDMAWRITE Case 5: Operation = CONTROL_OP_RDMAREAD *Receiver: Expect VIP_STATUS_DONE + VIP_STATUS_FORMAT_ERROR in status plus* Unreliable delivery: *VI still in connected state.* Reliable Delivery: *Sender: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* Reliable Reception: *Sender: either VIP_STATUS_REMOTE_DESC_ERROR or VIP_STATUS_TRANSPORT_ERROR or VIP_DESC_FLUSHED_ERROR in status, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.*	*AS: 5.1* *AS: 10.2* / *DG:5.2* *AS: 9.10.2* *AS: 2.5.2* *AS: 2.5.3*	
async	2	**ErrHRcvLen**	Valid Send with remote Receive Descriptor too small to hold data Unreliable delivery: *Sender: expect no notification of error, VI remains connected* *Receiver: expect VipRecvWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_LENGTH_ERROR in the status, VI remains connected.* Reliable Delivery: Sender: expect send to complete successfully, connection later broken with *error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: expect VipRecvWait() to complete in error as in Unreliable,*	AS:9.9.1 / DG:3.8 *AS:2.5.1* *AS:2.5.2*	

			RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), *VI in error state, queues flushed.*		
			Reliable Reception: *Sender: expect VipSendWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_REMOTE_DESC_ERROR in the status.* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).* *VI in error state, queues flushed.* *Receiver: same as Reliable Delivery.*	*AS:2.5.3*	
async	2	**ErrHRcvLen2**	VipPostRecv() with Length Errors where Segment Count > MaxSegmentsPerDescriptor *Expect VIP_STATUS_DONE + VIP_STATUS_LENGTH_ERROR in status plus* Unreliable delivery: *VI still in connected state.* Reliable Delivery: *Sender: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* Reliable Reception: *Sender: either VIP_STATUS_REMOTE_DESC_ERROR or VIP_STATUS_TRANSPORT_ERROR or VIP_DESC_FLUSHED_ERROR in status, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.*	AS:9.6.4 *AS:10.2* *AS:2.5.1* *AS:2.5.2* *AS:2.5.3* *AS:10.2*	/ DG:3.5.4 / *DG:5.2* / *DG:5.2*
async	2	**ErrHRcvPost1**	CATASTROPHIC ERROR: Recv Queue Post Descriptor Error *Case 1: Virtual address of descriptor not valid (valid + 2*PageSize).* *Case 2: Virtual address of descriptor not valid (NULL addr).* *Case 3: Memory handle of descriptor not valid (valid + 16).* *Case 4: Memory handle of descriptor not valid (NULL handle).* *Case 5: Previously deregistered descriptor.* *Case 6: Descriptor memory registered with Ptag different from VI Ptag.* All Reliability modes: *Sender: error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* Receiver: error callback (VIP_ERROR_POST_DESC), VI in error state, queues flushed. RECOMMENDED: callback (VIP_ERROR_CONN_LOST). *Note that side with error must call VipDisconnect() and empty queues after error before proceeding.*	AS:9.1 *AS:9.9.1* AS:2.5 *AS:9.9.1*	+ DG:6.3.1 / *DG:3.8* / *DG:3.8* + *DG:7.5*

async	2	**ErrHRcvPost2**	CATASTROPHIC ERROR: Post Descriptor Error w/ Descriptor not on 64-byte boundary. *Expect same results as ErrHRcvPost1 above.*	AS:9.6.1 *AS:9.9.1*	 + *DG:3.8*
async	2	**ErrHRcvProt**	VipPostRecv() with unregistered Data address in VIP data segment Unreliable delivery: *Case 1: Invalid Memory Handle in Data Segment (Handle = -1)* *Case 2: Unregistered Data Buffer.* Unreliable delivery: *Sender: expect no notification of error, VI remains connected* *Receiver: expect VipRecvWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_PROTECTION_ERROR in the status, VI remains connected.* Reliable Delivery: Sender: expect send to complete successfully, connection later broken with *error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: expect VipRecvWait() to complete in error as in Unreliable, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* Reliable Reception: *Sender: either VIP_STATUS_REMOTE_DESC_ERROR or VIP_STATUS_TRANSPORT_ERROR or VIP_DESC_FLUSHED_ERROR in status, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.*	AS:9.6.4 *AS:2.5.1* *AS:2.5.1* *AS:2.5.2* *AS:2.5.3*	/ DG:3.5.4
async	2	**ErrHRcvProt2**	VipPostRecv() with Ptag Protection Errors *Case 1: Data buffer memory registered with Ptag different from VI Ptag.* *Case 2: Descriptor memory registered with Ptag different from VI Ptag.* Unreliable delivery: *Sender: expect no notification of error, VI remains connected* *Receiver: expect VipRecvWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_PROTECTION_ERROR in the status, VI remains connected.* Reliable Delivery: Sender: expect send to complete successfully, connection later broken with *error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: expect VipRecvWait() to complete in error as in Unreliable, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.*	AS:9.6.4 AS:3.2.2.1 AS:3.2.2.1 *AS:2.5.1* *AS:9.10.2* *AS:2.5.2*	/ DG:3.5.4 / *DG:5.2*

			Reliable Reception: *Sender: expect VipSendWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_REMOTE_DESC_ERROR in the status. RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: same as Reliable Delivery.*	*AS:2.5.3*
async	2	**ErrHRdrUD**	VipPostSend() RDMA Read on Unreliable Delivery Connection without RdmaRead permissions. *Sender: Expect VIP_STATUS_FORMAT_ERROR in the status, VI still in connected state.* Receiver: No notification of error, VI still in connected state.	AS:9.6.1
async	2	**ErrHRdwForm**	VipPostSend() with invalid Control Field (RDMA Write w/o Immediate Data) *Cases 1-3: Same as ErrHSndForm above.* *Case 4: Address Segment Reserved != 0* *Expect same results as ErrHSndForm.*	AS:9.6.1
async	2	**ErrHRdwiForm**	VipPostSend() RDMA Write w/ Immediate Data Op with Descriptor Format Error *Cases 1-3: Same as ErrHSndForm above.* *Case 4: Address Segment Reserved != 0* *Expect same results as ErrHSndForm.*	AS:9.6.1
async	2	**ErrHRdwiLen**	VipPostSend() RDMA Write w/ Immediate Data Op with invalid length *Cases 1-5: Same as ErrHSndLen.* *Expect same as ErrHSndLen.*	AS:9.6.1
async	2	**ErrHRdwiProt1**	VipPostSend() RDMA Write w/ Immediate Data Op with Protection Error *Cases 1-2: Same as ErrHSndProt1.* *Expect same results as ErrHSndProt1.*	AS:9.6.1
async	2	**ErrHRdwiProt2**	VipPostSend() RDMA Write w/ Immediate Data Op with Ptag-Related Protection Error *Case1: Same as ErrHSndProt2.* *Expect same results as ErrHSndProt2.*	AS:9.6.1
async	2	**ErrHRdwiProt3**	RDMA Write w/ Immediate Data with Protection Error. (Detected on remote) *Case 1: Write to VI without RDMA Write permission* *Case 2: Write to remote memory that does not have RDMA write permissions* Unreliable delivery: Sender: No errors reported. VIs in connected state.	AS:9.9 / DG:3.8 *AS:2.5.1*

			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, VIs remain conn.		
			Reliable Delivery:	*AS:2.5.2*	
			Sender: expect write to complete successfully, connection later broken with error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.		
			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST). Vi in error state, queues flushed.	*AS:10.2*	*/ DG:5.2*
			Reliable Reception:	*AS:2.5.3*	
			Sender: expect write desc to complete w/ descriptor status VIP_STATUS_DONE and VIP_STATUS_RDMA_PROT_ERROR, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST). Vi in error state, queues flushed.	*AS:10.2*	*/ DG:5.2*
			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, VI in error state, queues flushed. RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).		
async	2	**ErrHRdwiProt4**	RDMA Write w/ Immediate Data with Protection Error. (Detected on remote)	AS:9.9	/ DG:3.8
			Case 1: RDMA Write with invalid remote buffer virtual address.		
			Case 2: RDMA Write with invalid remote memory handle (valid+1)		
			Case 3: Remote buffer registered w/ different ptag than remote VI.		
			Unreliable delivery:	*AS:2.5.1*	
			Sender: No errors reported. VIs in connected state.		
			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, VIs remain conn.		
			Reliable Delivery:	*AS:2.5.2*	
			Sender: expect write to complete successfully, connection later broken with error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.		
			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) Vi in error state, queues flushed.	*AS:10.2*	*/ DG:5.2*
			Reliable Reception:	*AS:2.5.3*	
			Sender: expect write desc to complete w/ descriptor status VIP_STATUS_DONE and VIP_STATUS_RDMA_PROT_ERROR, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) Vi in error state, queues flushed.	*AS:10.2*	*/ DG:5.2*
			Receiver: desc. status contains VIP_STATUS_PROTECTION_ERROR, VI in error state, queues flushed. RECOMMENDED: error callback (VIP_ERROR_CONN_LOST)		
async	2	**ErrHRdwiRcvQEmpty**	RDMA Write w/ Immediate Data with remote Receive Queue Empty error	AS:9.9.1	/ DG:3.8.1

Same expectations as ErrHSndRcvQEmpty.

async	2	**ErrHRdwLen**	VipPostSend() RDMA Write w/o Immediate Data Op with invalid length *Cases 1-5: Same as ErrHSndLen.* *Expect same as ErrHSndLen but:* *Receiver: VIP_STATUS_PARTIAL_ERROR not accepted.*	AS:9.6.1	
async	2	**ErrHRdwProt1**	VipPostSend() RDMA Write w/o Immediate Data Op with Protection Error *Cases 1-2: Same as ErrHSndProt1.* *Expect same results as ErrHSndProt1 but:* *Receiver: VIP_STATUS_PARTIAL_ERROR not accepted.* Reliable Delivery/Reception: *Receiver: One or both of error callback (VIP_ERROR_RDMAW_ABORT) and error callback(VIP_ERROR_CONN_LOST).* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) if no async error callback VIP_ERROR_RDMAW_ABORT received.*	AS:9.6.1	
async	2	**ErrHRdwProt2**	VipPostSend() RDMA Write w/o Immediate Data Op with Ptag-Related Protection Error *Case1: Same as ErrHSndProt2.* *Expect same results as ErrHSndProt2.*	AS:9.6.1	
async	2	**ErrHRdwProt3**	RDMA Write w/o Immediate Data with Protection Error. (Detected on remote) *Case 1: Write to VI without RDMA Write permission* *Case 2: Write to remote memory that does not have RDMA write permissions* Unreliable delivery: *No errors reported. VIs in connected state.* Reliable Delivery: *Sender: expect write to complete successfully, connection later broken with error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: error callback (VIP_ERROR_RDMAW_PROT),* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).* *VI in error state, queues flushed.* Reliable Reception: *Sender: expect write desc to complete w/ descriptor status VIP_STATUS_DONE and VIP_STATUS_RDMA_PROT_ERROR,* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).* *Vi in error state, queues flushed.* *Receiver: one or both of: error callback (VIP_ERROR_RDMAW_PROT) and error callback (VIP_ERROR_CONN_LOST), along with VI in error state, queues flushed.*	AS:9.9 *AS:2.5.1* *AS:2.5.2* *AS:9.9.1* *AS:2.5.3* *AS:10.2*	/ DG:3.8 / *DG:3.8.1* / *DG:5.2*

async	2	**ErrHRdwProt4**	RDMA Write w/o Immediate Data with Protection Error. (Detected on remote) *Case 1: RDMA Write with invalid remote buffer virtual address (addr + 2*PageSize)* *Case 2: RDMA Write with invalid remote memory handle (valid+1)* *Case 3: Remote buffer registered w/ different ptag than remote VI.* Unreliable delivery: *No errors reported. VIs in connected state.* Reliable Delivery: *Sender: expect write to complete successfully, connection later broken with error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: error callback (VIP_ERROR_RDMAW_PROT), RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) VI in error state, queues flushed.* Reliable Reception: *Sender: expect write desc to complete w/ descriptor status VIP_STATUS_DONE and VIP_STATUS_RDMA_PROT_ERROR, RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) Vi in error state, queues flushed.* *Receiver: one or both of: error callback (VIP_ERROR_RDMAW_PROT) and error callback (VIP_ERROR_CONN_LOST), along with VI in error state, queues flushed.*	AS:9.9 *AS:2.5.1* *AS:2.5.2* *AS:9.9.1* *AS:2.5.3* *AS:10.2*	/ DG:3.8 */ DG:3.8.1* */ DG:5.2*
async	2	**ErrHSndForm**	VipPostSend() Send Operation with Descriptor Format Error. *Case 1: Op = VIP_CONTROL_OP_RESERVED (invalid operation)* *Case 2: Reserved != 0 in VIP control segment* *Case 3: Control Field bits 15:4 != 0* *Sender: Expect VIP_STATUS_DONE + VIP_STATUS_FORMAT_ERROR in status plus* Unreliable delivery: *VI still in connected state.* Reliable Delivery/Reception: *Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.*	AS:9.6.1 *AS:10.2* *AS:9.10.2* *AS:2.5.[2,3]*	*/ DG:5.2*
async	2	**ErrHSndLen**	VipPostSend() Send Operation with invalid Length *Case 1: Control Segment Length = -1* *Case 2: Control Segment Length > MaxTransferSize for VI* *Case 3: Control Segment Length != sum of all LocalBufferLengths* *Case 4: Data Segment Length = -1*	AS:9.6.1 *AS:10.2*	*/ DG:5.2*

			Case 5: Segment Count > MaxSegmentsPerDesc *Sender: Expect VIP_STATUS_DONE + VIP_STATUS_LENGTH_ERROR + possible VIP_STATUS_PARTIAL_ERROR in status plus:* Unreliable delivery: Sender: VI still in connected state. *Receiver: ACCEPTED: Possible VIP_STATUS_PARTIAL_ERROR in descriptor status. VI still in connected state.* Reliable Delivery/Reception: *Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: ACCEPTED: VIP_STATUS_PARTIAL_ERROR in descriptor status, REQUIRED: error callback (VIP_ERROR_CONN_LOST) if no partial error in desc. -or- RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) if partial error in desc. VI in error state, queues flushed.*	*AS:9.10.2* *AS:2.5.[2,3]*	
async	2	**ErrHSndLen2**	VipPostSend() Send Operation with Length > ViAttr.MTS but less than Nic.MTS *Set ViAttr.MTS = Nic.MTS / 2. If VipCreateVI() returns VIP_INVALID_MTU, then verify and test exits. Otherwise, post send desc. with length > ViAttr.MTS < Nic.MTS and expect:* *Sender: Expect VIP_STATUS_DONE + VIP_STATUS_LENGTH_ERROR + possible VIP_STATUS_PARTIAL_ERROR in status plus:* Unreliable delivery: Sender: VI still in connected state. *Receiver: ACCEPTED: Possible VIP_STATUS_PARTIAL_ERROR in descriptor status. VI still in connected state.* Reliable Delivery/Reception: *Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.* *Receiver: ACCEPTED: VIP_STATUS_PARTIAL_ERROR in descriptor status, REQUIRED: error callback (VIP_ERROR_CONN_LOST) if no partial error in desc. -or- RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) if partial error in desc. VI in error state, queues flushed.*	AS:9.10.[4,5] *AS:9.10.2* *AS: 2.5.1* *AS: 2.5.[2,3]*	DG:4.[5,6] *DG:3.2.1* *DG:5.2* *DG:5.2* *DG:5.2*
async	2	**ErrHSndPost1**	CATASTROPHIC ERROR: Send Post Descriptor Error *Case 1: Virtual address of descriptor not valid (valid + 2*PageSize).* *Case 2: Virtual address of descriptor not valid (NULL addr).* *Case 3: Memory handle of descriptor not valid (valid + 16).* *Case 4: Memory handle of descriptor not valid (NULL handle).* *Case 5: Previously deregistered descriptor.*	AS:9.1 *AS:9.9.1*	+ DG:6.3.1 / *DG:3.8*

			Case 6: Descriptor memory registered with Ptag different from VI Ptag.	AS:3.2.2.1	
			All Reliability modes:	AS:2.5	
			Sender: error callback (VIP_ERROR_POST_DESC), VI in error state, queues flushed.	*AS:9.9.1*	/ *DG:3.8*
			RECOMMENDED: callback (VIP_ERROR_CONN_LOST).		
			Receiver: error callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.		
			Note that side with error must call VipDisconnect() and empty queues after error before proceeding.		+ *DG:7.5*
async	2	**ErrHSndPost2**	CATASTROPHIC ERROR: Post Descriptor Error w/ Descriptor not on 64-byte boundary.	AS:9.6.1	
			Expect same results as ErrHSndPost1 above.	*AS:9.9.1*	+ *DG:3.8*
async	2	**ErrHSndProt1**	VipPostSend() Send Operation with Protection Error	AS:9.6.1	
			Case 1: Invalid Memory Handle in Data Segment (Handle = -1)	*AS:10.2*	/ *DG:5.2*
			Case 2: Invalid Data Buffer Virtual Address (Addr = -1)		
			Sender: Expect VIP_STATUS_DONE + VIP_STATUS_PROTECTION_ERROR + possible VIP_STATUS_PARTIAL_ERROR in status plus	*AS:9.10.2*	
			Unreliable delivery:		
			Sender: VI still in connected state.		
			Receiver: ACCEPTED: VIP_STATUS_PARTIAL_ERROR in recv desc. VI still in connected state.		
			Reliable Delivery/Reception:		
			Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.		
			Receiver: ACCEPTED: VIP_STATUS_PARTIAL_ERROR in descriptor status, REQUIRED: error callback (VIP_ERROR_CONN_LOST) if no partial error in desc. -or- RECOMMENDED: error callback (VIP_ERROR_CONN_LOST) if partial error in desc. VI in error state, queues flushed.		
async	2	**ErrHSndProt2**	VipPostSend() Send Operation with Ptag-Related Protection Error	AS:9.6.1	
			Case 1: Data buffer memory registered with Ptag different from VI Ptag.	AS:3.2.2.1	
			Sender: Expect VIP_STATUS_DONE + VIP_STATUS_PROTECTION_ERROR in status plus	*AS:9.10.2*	/ *DG:5.2*
			Unreliable delivery:	*AS:2.5.1*	
			VI still in connected state.		
			Reliable Delivery/Reception:	*AS:2.5.[23]*	
			Sender: RECOMMENDED: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.		
			Receiver: error callback (VIP_ERROR_CONN_LOST), VI in error state, all outstanding request descriptors completed in error on both queues.		

async	2	**ErrHSndRcvQEmpty**	Valid Send with remote Receive Queue Empty error Unreliable delivery: *Expect no notification on either side, VIs remain connected.* Reliable Delivery: Sender: expect send to complete successfully, connection later broken with error *callback (VIP_ERROR_CONN_LOST), VI in error state, queues flushed.* *Receiver: expect callback (VIP_ERROR_RECVQ_EMPTY).* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).* *VI in error state.* Reliable Reception: *Sender: expect VipSendWait() to complete in error (VIP_ERROR_DESCRIPTOR) with VIP_STATUS_REMOTE_DESC_ERROR in the status.* *RECOMMENDED: error callback (VIP_ERROR_CONN_LOST).* *VI in error state, queues flushed.* *Receiver: same as Reliable Delivery.*	AS:9.9.1 *AS:2.5.1* *AS:2.5.2* *AS:2.5.3*	/ DG:3.8.1
error	2	**ErrorCallbackErr**	VipErrorCallback() with invalid NicHandle *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER.*	AS:9.9.1	/ DG:3.8.1

async.xls

B.9. Name Service

Type	TL	Name	Description	Section
error	2	**NSAddrSvcErr**	Call VipNSGetHostByAddr() with invalid addresses *case 1: Nameservice was not initialized.* *case 2: Address does not exist.* *case 3: Empty address.* *Expect VIP_ERROR_NAMESERVICE*	DG:3.9.3
error	2	**NSAddrParamErr**	VipNSGetHostByAddr() with invalid parameters *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *case 4: Nameservice was shutdown.* *Expect VIP_INVALID_PARAMETER*	DG:3.9.3
basic	2	**NSGet**	VipNSGetHostByAddr() functional test. *obtain the name of every configured NIC (from bin\netaddr file).* *call with NameLen = 0, expect VIP_INVALID_PARAMETER, correct NameLen returned* *call with correct NameLen, expect VIP_SUCCESS.* *(names must be manually verified).*	DG:3.9.3
			VipNSGetHostByName() functional test. *use names obtained from VipNSGetHostByAddr() above.* *Expect VIP_SUCCESS, with correct address returned.*	DG:3.9.2
error	2	**NSInitErr**	VipNSInit() with invalid parameters *case 1: Previously closed NicHandle. ** *case 2: NULL NicHandle. ** *case 3: Random NicHandle. ** *Expect VIP_INVALID_PARAMETER*	DG:3.9.1
error	2	**NSNameSvcErr**	Call VipNSGetHostByName() with invalid names. *case 1: Nameservice was not initialized.* *case 2: Name does not exist.* *case 3: NULL string.* *Expect VIP_ERROR_NAMESERVICE*	DG:3.9.2
error	2	**NSNameParamErr**	VipNSGetHostByName() with invalid parameters	DG:3.9.2

			*case 1: Previously closed NicHandle. * case 2: NULL NicHandle. * case 3: Random NicHandle. * case 4: Nameservice was shutdown. case 5: Invalid NameIndex (huge) case 6: HostAddress field too small. Expect VIP_INVALID_PARAMETER*	
error	2	**NSShutdownErr**	VipNSShutdown() with invalid parameters *case 1: Previously closed NicHandle. * case 2: NULL NicHandle. * case 3: Random NicHandle. * Expect VIP_INVALID_PARAMETER*	DG:3.9.4

names.xls